Seminarausarbeitung zu Xen

im Proseminar Konzepte von Betriebssystemkomponenten

Johannes Schlumberger
spjsschl@cip.informatik.uni-erlangen.de

Kurzzusammenfassung

Dieses Dokument soll einen Überblick über Ziele und Design von Xen geben und dabei Grundkonzepte der Virtualisierung im Allgemeinen und der von Xen geprägten Paravirtualisierung verdeutlichen.

1 Einführung

1.1 Virtualisierung

Es ist schwer den Begriff Virtualisierung prägnant und gleichzeitig aussagekräftig zu definieren. Unter einer virtuellen Maschine versteht man eine in Software implementierte Maschine, die ihren Anwendungen eine Abstraktionsschicht bietet. Diese Abstraktionsschicht ist in der klassischen Virtualisierung meist volltransparent.

Erste Erfahrungen mit dem Konzept der Virtualisierung wurden bereits 1972 von IBM mit dem VM/370 gesammelt. Auf der nackten Hardware lief ein Betriebssystem, gennant VM370, das verschiedene virtuelle VM/370 Maschinen multiplexen konnte.

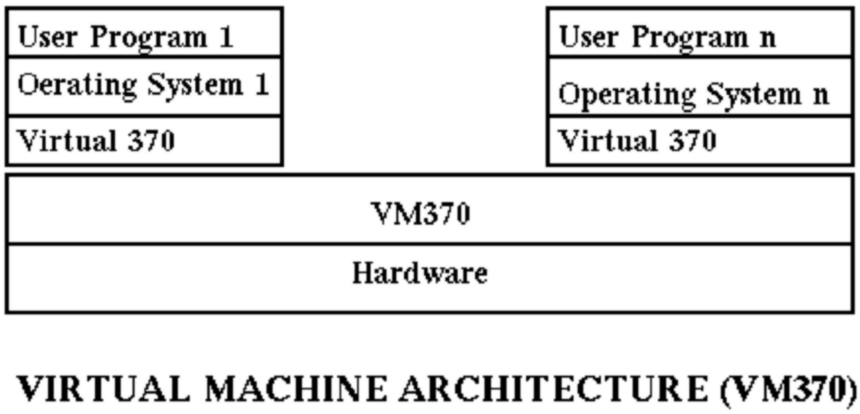

Abbildung 1: Konzept der IBM VM/370

Vor allem zwei Vorteile der Virtualisierungstechnik sind sofort erkennbar; erstens wird durch den Ablauf eines Betriebssystems in einer Art Gefängnis (engl. jail) die Sicherheit erhöht. Gelingt es einem Angreifer aus dem Gastbetriebssystem auszubrechen, hat er noch

immer keinen Zugriff auf die nackte Hardware und auf Anwendungen, die in anderen Gastbetriebssystemen laufen. Zweitens ist es möglich Betriebsmittel den verschiedenen Gastbetriebssystemen sehr flexibel zuzuteilen, was sich vor allem anbietet wenn die einzelnen Systeme starken Schwankungen in ihrer Auslastung unterworfen sind.

Der Hauptnachteil der Virtualisierung liegt in den starken Performanzverlusten, die gegenüber nichtvirtualisierten Maschinen auftreten. Aus meiner eigenen Erfahrung kann ich sagen, dass bei Vollvirtualisierung Verluste von 40-60% die Regel sind.

2 Xen

Xen ist eine virtuelle Maschine, die speziell für den Einsatz auf der x86-Architektur entwickelt wurde. Sie ist ausgelegt auf gängige Betriebssysteme wie Linux, *BSD oder Windows XP. Xen ist sehr sicher und verfügt über gute Mechanismen zur Ressourcenisolation.

Ressourcenisolation ist wichtig, um zu vermeiden, dass einzelne Gastbetriebssysteme die Hardware durch Effekte wie zum Beispiel Seitenflattern (engl. trashing) oder ähnliches so stark ausbremsen, dass andere Gastbetriebssysteme in Mitleidenschaft gezogen werden. Xen ist funktional nicht wesentlich schwächer oder langsamer als nichtvirtualisierte Maschinen.

2.1 Paravirtualisierung

Um diesen Anforderungen gerecht zu werden bedient sich Xen den Mitteln der Paravirtualisierung oder Teilvirtualisierung. Im Gegensatz zur Vollvirtualisierung, bei der die virtuelle Hardware genau der echten Hardware entspricht, wird bei der Paravirtualisierung die echte Hardware nicht bis ins Detail nachgebildet. Das hat zur Folge, dass bei paravirtualisierten Systemen, im Gegensatz zu Vollvirtualisierten, Änderungen an den Gastbetriebssystemen nötig sind.

Nimmt man diese Änderungen aber in Kauf hat die Paravirtualisierung einige bestechende Vorteile zu bieten. Es ist nicht nur möglich etliche Schwierigkeiten, die gerade bei dem Versuch die x86-Architektur zu virtualisieren auftreten, elegant zu umgehen, man kann den Gastbetriebssystemen neben dem Zugriff auf Virtuelle auch den Zugriff auf echte Ressourcen ermöglichen (z.B. Zeit oder Speicheradressen), was eine enorme Erleichterung zum Beispiel für die Einhaltung von Timeouts in der Kommunikation mit der Außenwelt oder beim Speicherzugriff bedeutet.

Der Ansatz der Paravirtualisierung selbst ist zwar älter als Xen, aber das Xen-Projekt hat ihn entscheidend geprägt, und stellt praktisch die Referenzimplementierung dar.

2.2 Designentscheidungen und Ziele

Fünf Ziele waren maßgeblich beim Design von Xen.

1. ABI darf nicht verändert werden
2. keine Einschränkungen im Mehrbenutzerbetrieb
3. hohe Performanz
4. starke Ressourcenisolation
5. Paravirtualisierung soll Performanzverluste auffangen

Würde man die ABI (Application binary Interface), also die binäre Schnittstelle für Anwendungen verändern, würde das bedeuten, dass alle Anwendungen, die auf Xen laufen sollen extra angepaßt werden müssten. Das wäre der Akzeptanz und Bedienbarkeit von Xen sehr abträglich gewesen. Hohe Performanz und die damit einhergehende Ressourcenisolation waren erklärtes Ziel von Xen. Die bei Vollvirtualisierung auftretenden inakzeptabel hohen Verluste sollten mit Paravirtualisierung aufgefangen werden.

2.3 Umsetzung

2.3.1 Speicher

Die x86-Architektur verfügt über keinen softwaregesteuerten TLB; TLB-Fehlschläge werden automatisch vom Prozessor über Hardware-Seitentabellen verarbeitet. Das macht eine Virtualisierung von x86-Hardware teuer und aufwändig. Deshalb ist es nötig, wenn man performant virtualisieren will, alle gültigen Adressen in den Hardware-Seitentabellen bereitzuhalten.

Das Xen-Entwicklerteam hat hier zwei Designentscheidungen getroffen. Erstens sollen Gastbetriebssysteme ihre Seitentabellen soweit irgend möglich selbst verwalten, während Xen sich ausschließlich um Isolation und Sicherheit kümmert. Zweitens residiert Xen fest in den oberen 64 MB Hauptspeicher um einen kompletten TLB-Flush bei jedem Kontextwechsel zu vermeiden.

Das stellt keine Schwierigkeit dar, da die oberen 64 MB Hauptspeicher sowieso von keinem gängigen x86-Betriebssystem genutzt werden.

Wenn ein Gastbetriebssystem eine neue Seite anfordert, allokiert und initialisiert es diese selbst aus seinem eigenen, ihm vom Xen zugeteilten Speicher. Anschliessend macht es diese neue Seite Xen bekannt und gibt gleichzeitig die Schreibrechte für Seitentabellen auf. Alle folgenden Updateanfragen werden von Xen erst validiert bevor sie abgearbeitet werden. Xen verbietet den Gastbetriebssystemen dabei generell in die Seitentabellen zu schreiben, in Seitentabellen anderer Gastbetriebssysteme Einsicht zu nehmen oder in die oberen 64 MB Speicher zu greifen, die für Xen reserviert sind.

Die Gastbetriebssysteme haben die Möglichkeit Seitenupdates zu sogenannten Batchupdates zusammenzufassen, die von Xen gesammelt abgearbeitet werden, um die Kosten für den Kontextwechsel zu minimieren.

Segmentierung funktioniert ähnlich. Xen kontrolliert Anfragen für die Segmentdeskriptortabelle, und verbietet alle, die Xens Privilegienlevel erreichen oder versuchen Teile von Xens Speicher zu inkludieren.

2.3.2 Prozessor

Der x86-Prozessor verfügt über vier Sicherheitsringe in Hardware (Ring0 - Ring3). Üblicherweise läuft das Betriebssystem im einzigen privilegierten Ring, Ring0, Anwendungen in Ring3. Die Ringe 1 und 2 sind üblicherweise ungenutzt (Ausnahme: OS/2).

Indem man das Gastbetriebssystem so modifiziert, dass es in Ring1 läuft erreicht man einerseits eine sichere Isolation zwischen Xen und dem Gastbetriebssystem und andererseits auch eine sichere Isolation zwischen dem Gastbetriebssystem und seinen Anwendungen. Jeder Versuch eines Gastbetriebssystems, einen privilegierten Befehl auszuführen wird vom Prozessor entweder stillschweigend ignoriert oder mit einem Fehler quittiert.

Privilegierte Anweisungen müssen also von Xen validiert und paravirtualisiert ausgeführt werden.

Für Ausnahmen wird in Xen eine Tabelle mit einem Eintrag für jeden Exceptionhandler bekannt gemacht. Diese Handler können unveränderte x86-Handler sein, da die Exceptionstack-Frames von Xen nicht verändert werden. Eine Ausnahme stellt lediglich der Handler für Seitenfehler dar. Bei einem Seitenfehler steht in der x86-Architektur die fehlerhafte Adresse im privilegierten Register `CR2`, das vom Gastbetriebssystem nicht lesbar ist. Der Inhalt dieses Registers wird von Xen in einem erweiterten Stackframe an das Gastbetriebssystem weitergereicht, was natürlich eine Anpassung des entsprechenden Handlers erfordert. Generell werden Ausnahmen von Xen behandelt, indem erst der Exceptionstack von Xen in das Gastbetriebssystem kopiert wird und anschließend die Kontrolle an den entsprechenden Handler übergeben wird, der die Ausnahme dann behandelt.

Indem Gastbetriebssysteme ihre Handler von Xen validieren lassen, ehe sie in die Handlertabelle eingetragen werden, erlaubt Xen diese ohne Indirektion über Ring0 direkt anzuspringen. Der Handler darf um valide zu sein natürlich nicht verlangen in Ring0 zu laufen. Diese sogenannten "fast handler" verbessern die Performanz bei Systemaufrufen erheblich.

2.3.3 E/A

Xen virtualisiert im Gegensatz zu vollvirtualisierten Produkten existierende Hardware nicht. Statt dessen stellt Xen eine neue Abstraktionsschicht für Geräte zur Verfügung, praktisch virtuelle Xen-Geräte. Die Gerätetreiber dafür müssen dem Gastbetriebssystem entsprechend hinzugefügt werden. Das Gastbetriebssystem arbeitet dann nur noch mit den Xenspezifischen Treibern auf den Xen-Geräten.

E/A-Daten werden vertikal über shared-memory Segmente ausgetauscht, was einerseits hochperformant ist, andererseits es Xen ermöglicht die E/A-Daten einer Kontrolle zu unterziehen.

Hardware-Unterbrechungen werden durch ein leichtgewichtiges Ereignissystem nachgebildet. Jedes Gastbetriebssystem verfügt über eine Bitmap in der jedem ausstehenden Ereignis ein Bit zugeordnet ist, das bei Eintreffen des Ereignisses von Xen gesetzt wird. Optional ruft Xen zusätzlich einen vorher vom Gastbetriebssystem festgelegten Handler auf, um das Ereignis zu prozessieren.

Diese Handler können von den Gastbetriebssystemen gesperrt werden um Extrakosten durch wiederholtes Aufwecken bei Ereigniseintritten zu sparen.

2.3.4 Zeit

Xen stellt über sein Ereignissystem drei Arten von Zeit zur Verfügung. Reale Zeit wird gemessen ab Nanosekunden seit dem Urladen von Xen, virtuelle Zeit wird für jede Domäne getrennt gemessen und schreitet nur voran wenn die aktuelle Domäne läuft, Uhrzeit ist die reale Zeit mit einem Offset.

Dadurch dass Xen diese drei Arten von Zeit gleichzeitig zur Verfügung stellt, ist es den Gastbetriebssystemen möglich jeweils die Zeit zu verwenden, die für die aktuelle Aufgabe sinnvoll ist. Zum Beispiel reale Zeit für TCP-Timeouts, virtuelle Zeit um seinen internen Scheduler seinen Anwendungen korrekte Zeitscheiben zuteilen zu lassen und die Uhrzeit um die Desktopuhr eines Benutzers richtig darzustellen.

3 Evaluation

3.1 Portabilität

Um ein Betriebssystem auf Xen zu portieren muss im Wesentlichen sein x86-abhängiger Code portiert, die Xen-Gerätetreiber geschrieben und einige andere kleinere Dinge geändert werden. Der Aufwand um Linux nach Xen zu portieren waren etwa 1400 Codezeilen (etwa 1,4%) des Linuxcodes. Bei WindowsXP ist der Aufwand etwas grösser, bei BSD etwa in derselben Grössenordnung wie bei Linux.

3.2 Relative Performanz

Das folgende Diagramm zeigt eine Reihe von Tests der verschiedensten Art (Kompiliertests, Prozessorperformanz, Dateisystemlast, usw.) ausgeführt auf nichtvirtualisiertem Linux (L), Xenolinux (X), VMware, einer vollvirtualisierten, kommerziellen Virtualisierunglösung (V) und UserModeLinux (U).

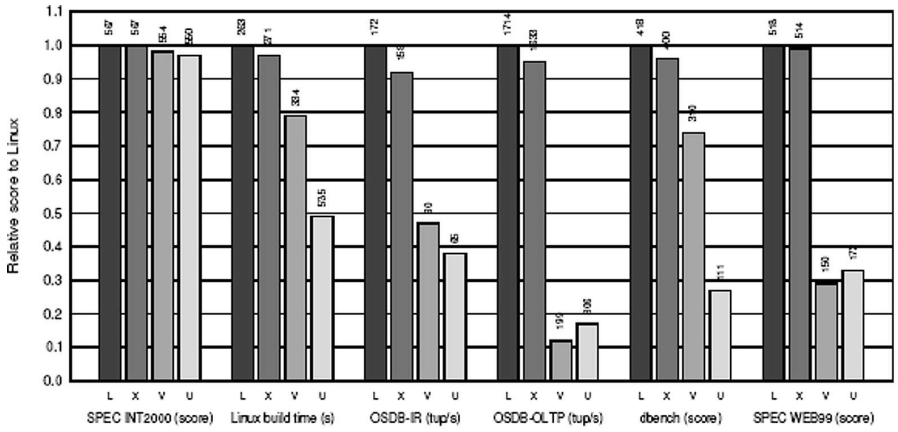

Figure 3: Relative performance of native Linux (L), XenoLinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).

Abbildung 2: XenoLinux im Vergleich mit UML, VMware und nativem Linux

Wie klar zu sehen ist, zeigt die Paravirtualisierung Wirkung. Ein Einbruch durch Virtualisierung ist auch bei Xen noch zu erkennen, aber er ist unvergleichbar viel kleiner als bei bisherigen Virtualisierungsprodukten.

3.3 Skalierbarkeit

Xen soll auf etwa 100 parallel laufende Domänen skalieren können. Eine Domäne mit Standard-Redhat-Server-Installation hat einen Speicherverbrauch von etwa 6,4 MB ohne Swap. Durch Benutzung einer Swappartition oder -datei kann dieser Bedarf um weitere 2 MB auf 4,2 MB gesenkt werden.

Der Speicherplatzbedarf skaliert linear, so dass mit 100 Domänen etwa 600 MB Hauptspeicher verbraucht werden. Dies stellt für moderne Serverhardware keine Herausforderung dar.

3.4 Latenz

Um die Verfügbarkeit von Domänen in Xen zu prüfen, wurde die mittlere Rundlaufzeit eines UDP-Paketes in zwei Szenarien untersucht. Im ersten Szenario laufen gleichzeitig 128 Domänen unter Vollast. Hier beträgt die durchschnittliche Antwortzeit 147 ms. Im zweiten Szenario kommt eine weitere Domäne hinzu, die aber nicht belastet wird. Jetzt beträgt die mittlere Antwortzeit über die unbelastete Domäne 5,4 ms. Ein Hinweis darauf, dass die schwer zu messende Effektivität von Ressourcenisolationsmechanismen hier vorhanden zu sein scheint.

Verkleinert man die Zeitscheiben des Xenschedulers für die einzelnen Gastbetriebssysteme, so werden die Antwortzeiten sogar noch verbessert.

3.5 Zusammenfassung

Den Xen-Entwicklern ist es offensichtlich gelungen, durch eine Reihe gut getroffener Designentscheidungen ein Virtualisierungssystem zu schaffen dass sich duch sehr hohe Performanz, gute Ressourcenisolation und gute Skalierbarkeit im Bereich bis etwa 100 Domänen auszeichnet.

4 Literaturverzeichnis

Literatur

[1] Paul Barham e.a.. Xen and the Art of Virtualization. Cambridge, 2003.