



Fachbereich Technik
Abteilung Elektrotechnik und Informatik

Bachelor-Thesis

Effiziente Speicherung virtueller Festplatten mit bestehender OpenSource-Software (Arbeitstitel)

Bastian de Groot

4. Januar 2011

Prüfer Prof. Dr. Jörg Thomaschewski

Zweitprüfer Dr. Arvid Requate

Inhaltsverzeichnis

1	Einleitung	4
1.1	Zieldefinition	5
1.2	Vorgehen und Kurzzusammenfassung	5
1.3	Anmerkung zur Verwendung von Open-Source	6
1.4	Anmerkung zu den verwendeten Literaturquellen	6
2	Analyse Copy-on-Write	8
2.1	Sparse-Dateien	9
2.2	qcow2	10
2.3	VHD	11
2.4	dm-snapshots	11
2.5	LVM-Snapshots	12
2.6	Benchmarks	13
2.6.1	Testbedingungen	13
2.6.2	Testergebnisse	14
2.7	Fazit	17
2.7.1	KVM	18
2.7.2	Xen	18
3	Analyse Verteilung von Images	19
3.1	Multicast	19
3.2	BitTorrent	21
3.3	NFS	22
3.4	Vergleich	23

3.4.1	Skalierbarkeit	24
3.4.2	Störanfälligkeit	24
3.4.3	Verteilungsdauer	25
3.5	Fazit	27
4	Synthese	28
4.1	Konzept	28
4.1.1	Steuerung und Kommunikation	28
4.1.2	Verteilung	29
4.1.3	Klonen	30
4.2	Realisierung einer Komplettlösung	30
4.2.1	Rahmenbedingungen	30
4.2.2	Steuerung und Kommunikation	31
4.2.3	Einrichtung eines Virtualisierungshosts	33
4.2.4	Verteilung	34
4.2.5	Klonen	35
4.2.6	Skripte/Programme	37
5	Zusammenfassung und Ausblick	39
5.1	Zusammenfassung	39
5.2	Fazit	39
5.3	Ausblick	39
6	Anhang	40

1 Einleitung

Von “Betriebssystemvirtualisierung” spricht man, wenn sich mehrere virtuelle Betriebssysteminstanzen Hardwareressourcen wie CPU, RAM oder Festplatten teilen. Der Virtualisierungskern (Hypervisor) stellt den virtuellen Betriebssysteminstanzen eine in Software und Hardware realisierte Umgebung zur Verfügung, die für die darin laufenden Instanzen kaum von einer echten Hardwareumgebung unterscheidbar sind. Es gibt unterschiedliche technische Ansätze der Virtualisierung, wie Paravirtualisierung oder Vollvirtualisierung. Diese Kategorisierung bezieht sich darauf, wie der Hypervisor die vorhandene Hardware für die virtuelle Instanz bereitstellt. Auf diesem Gebiet gibt es eine sehr aktive Entwicklung. [Prz] [Bau]

Wenig beachtet bei der Entwicklung von Virtualisierungssoftware ist jedoch die Speicherung von virtuellen Festplatten. In dieser Arbeit wird dieser Punkt aufgegriffen und die Möglichkeit der Optimierung mit der Copy-on-Write Strategie beleuchtet.

Copy-on-Write ist eine Optimierungsstrategie, die dazu dient unnötiges Kopieren zu vermeiden. Diese Strategie wird vom Linux-Kernel genutzt um Arbeitsspeicher einzusparen. Aber auch bei der Desktopvirtualisierung wird Copy-on-Write eingesetzt, um die benötigte Zeit für die Bereitstellung einer geklonten virtuellen Maschine minimieren. Hierbei wird nicht für jeden Benutzer ein eigenes Image kopiert, sondern alle Benutzer verwenden ein Master-Image. Falls ein Benutzer Änderungen an die-

sem Master-Image vornimmt, werden die Änderungen separat abgespeichert.

1.1 Zieldefinition

Ziel dieser Arbeit ist es, Möglichkeiten zur effizienten Speicherung von virtuellen Festplatten aufzuzeigen. Hierbei wird ausschließlich auf bestehende Open Source Lösungen zurückgegriffen (siehe 1.3). Die freien Open Source Lösungen werden miteinander verglichen und eine effiziente Lösung herausgearbeitet. Außerdem wird betrachtet, wie die für das Copy-on-Write benötigten Master-Images im Netzwerk effizient verteilt werden können.

1.2 Vorgehen und Kurzzusammenfassung

Zunächst werden die vorhandenen Softwarelösungen für Copy-on-Write und für die Verteilung der Master-Images erläutert. Danach werden diese anhand verschiedener anwendungsrelevanter Kriterien miteinander verglichen. Nachdem die besten Lösungen beider Kategorien gefunden wurden, werden Softwaretools erstellt, die die Nutzung der gefundenen Lösung ohne tiefgreifende Vorkenntnisse ermöglicht.

1.3 Anmerkung zur Verwendung von Open-Source

Für die Verwendung von Open-Source gibt es mehrere Gründe. Zum einen setzen führende Hersteller von Virtualisierungssoftware wie zum Beispiel Citrix aktuell auf Open-Source-Lösungen. Somit können in dieser Arbeit die neuesten Entwicklungen aufgezeigt werden. Zum anderen ist der Einsatz von proprietärer Software im finanziellen Rahmen dieser Arbeit nicht möglich. Der letzte wichtige Grund sind Lizenzprobleme bei proprietärer Software. Sie unterbinden zum Beispiel das Veröffentlichen von Performance-Tests oder die Distribution mit selbst erstellter Software [vmw].

1.4 Anmerkung zu den verwendeten Literaturquellen

Diese Arbeit bezieht sich neben den herkömmlichen Literaturquellen auch auf Mailinglisten- und Forenbeiträge, sowie Blogeinträge.

Bei Quellenangaben im Bereich der Open Source Software gibt es einige Punkte die zu beachten sind. Es gibt keine einheitliche Dokumentation der Software. Häufig sind die Informationen nicht an einer zentralen Stelle vereint, sondern liegen verstreut im Internet in Foren, Blogs, Mailinglisten oder auch in Manpages und den Quelltexten selbst. Die Relevanz und die Richtigkeit einer solcher Quellen ist schwer zu bewerten, da Blogs, Mailinglisten und Foren keinen Beschränkungen unterliegen.

Die oben genannte Verstreuung birgt, neben der schwierigen Bewertbar-

keit der Richtigkeit und Relevanz, ein weiteres Problem. Da sehr viele Autoren zum einem Thema etwas schreiben, werden unterschiedliche Begriffe synonym verwendet oder sind mehrdeutig.

Alle Quellen sind mit der zu Grunde liegenden Erfahrung des Autors dieser Arbeit ausgewählt und überprüft, können aber aus den oben genannten Gründen keine absolute Richtigkeit für sich beanspruchen.

2 Analyse Copy-on-Write

Für das Erstellen mehrerer gleichartiger Virtueller Maschinen benötigt man mehrere Virtuelle Festplatten. Das kann man auf herkömmliche Art und Weise lösen, in dem ein vorhandenes Festplattenimage n mal kopiert wird. Durch das häufige Kopieren entstehen allerdings große Mengen an Daten. Außerdem benötigt es viel Zeit Festplattenimages zu kopieren. Um diesen beiden Problemen entgegen zu wirken werden Copy-on-Write-Strategien eingesetzt.

Die Copy-on-Write-Strategie wird von Unix-artigen Betriebssystemen verwendet, um Arbeitsspeicher einzusparen. Es wird eingesetzt um nicht den ganzen Speicherbereich eines “geforkten” Prozesses kopieren zu müssen [CB05]. Die Vorteile der Optimierungsstrategie zeigen sich jedoch auch bei der Speicherung virtueller Festplatten.

Wie in Abbildung 2.1 schematisch dargestellt wird, wird bei Copy-on-Write nicht das gesamte Image kopiert. Es werden in dem Copy-on-Write-Image nur die Veränderungen zu dem so genannten Master- oder Quellimage gespeichert. Für die Platzersparnis werden Sparse-Dateien genutzt, welche im Folgenden erklärt werden. Außerdem werden die unterschiedlichen Verfahren zur Verwendung von Copy-on-Write erläutert und analysiert.

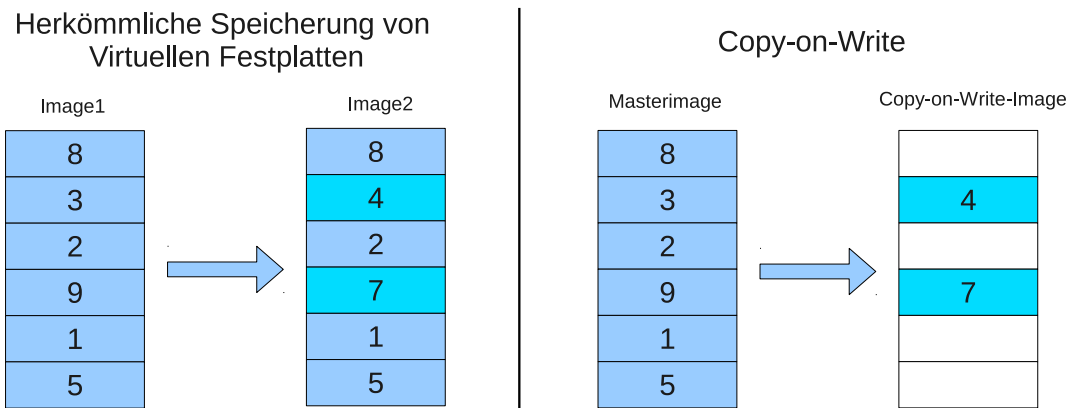


Abbildung 2.1: Copy-on-Write

2.1 Sparse-Dateien

Eine Sparse-Datei ist eine Datei, die nicht vom Anfang bis zum Ende beschrieben ist. Sie enthält also Lücken. Um Speicherplatz zu sparen, werden diese Lücken bei Sparse-Dateien nicht auf den Datenträger geschrieben. Die Abbildung 2.2 zeigt, dass der tatsächlich benutzte Speicherplatz auf der Festplatte weitaus geringer sein kann als die eigentliche Dateigröße [spa].

Eine Sparse-Datei ist kein eigenes Imageformat sondern eine Optimierungsstrategie. Sie verhilft Copy-on-Write-Images zu einer großen Platzersparnis. In Imageformaten wie qcow2 oder VHD ist diese Optimierungsstrategie ein fester Bestandteil.

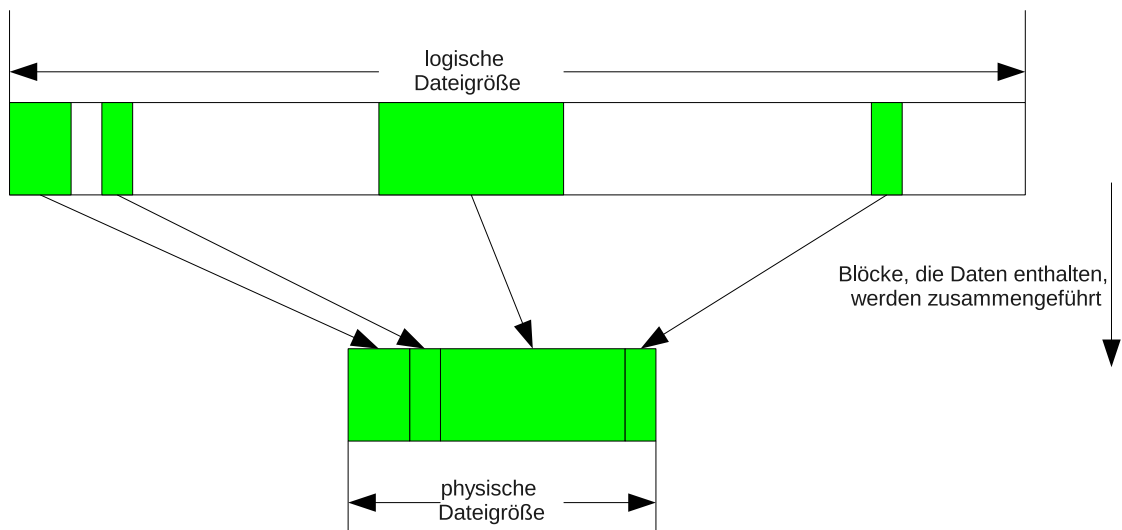


Abbildung 2.2: Sparse-Datei

2.2 qcow2

Das Imageformat qcow2 ist im Rahmen des qemu Projekts entwickelt wurde [qem]. Es ist der Nachfolger des ebenfalls aus dem qemu Projekt stammenden Formats qcow [McL].

Vorteile

- einfache Einrichtung
- aktive Entwicklung im Rahmen der Projekte KVM und qemu

Nachteile

- aktuell fehlende Unterstützung durch mit Xen und andere offene Virtualisierungstechniken (z.B. VirtualBox)

2.3 VHD

Das Format VHD ist von Conectix und Microsoft entwickelt worden. Die Spezifikation des Imageformats wurde von Microsoft im Zuge des “Microsoft Open Specification Promise” freigegeben [mso] [vhd]. Seit der Freigabe der Spezifikation bieten einige Open Source Virtualisierungslösungen wie qemu, Xen oder VirtualBox die Möglichkeit dieses Format zu verwenden.

Vorteile

- einfache Einrichtung
- Unterstützung durch Softwarehersteller mit hoher Marktakzeptanz

Nachteile

- Weiterentwicklung scheint derzeit fragwürdig
- Verwendung der Copy-on-Write-Funktion von VHD mit KVM nicht möglich

2.4 dm-snapshots

Die dm-snapshots sind eine Funktion des Device Mappers. Device Mapper ist ein Treiber im Linux-Kernel. Er erstellt virtuelle Gerätedateien, die mit bestimmten erweiterten Features wie zum Beispiel Verschlüsselung ausgestattet sind [Bro]. Bei dm-snapshots wird eine solche virtuelle Gerätedatei erstellt, die aus zwei anderen Gerätedateien zusammengesetzt wird. Die erste Gerätedatei ist der Ausgangspunkt, wenn daran Änderungen vorgenommen werden, werden sie als Differenz in der zwei-

ten Gerätedatei gespeichert [dmk].

Die von Device Mapper erstellten Gerätedateien benötigen keine Unterstützung der Virtualisierungstechnik, da sie für diese nicht von physikalischen Festplattenpartitionen unterscheidbar sind. Dieses ist nicht nur ein Vorteil, sondern zugleich auch ein Nachteil. Es muss immer vor dem Starten einer virtuellen Maschine das Copy-on-Write-Image und das Masterimage zu einer Gerätedatei verbunden werden.

Vorteile

- hohes Entwicklungsstadium
- gesicherte Weiterentwicklung
- unabhängig von Virtualisierungstechnik

Nachteile

- Aufwendige Einrichtung
- erfordert zusätzlichen Programmstart vor dem VM-Start

2.5 LVM-Snapshots

LVM-Snapshots sind ein Teil des Logical Volume Managers. LVM ist eine Software-Schicht die über den eigentlichen Hardware-Festplatten einzuordnen ist [lvma] [lvmc]. Sie basiert auf Device Mapper [lvmb]. LVM ermöglicht das Anlegen von virtuellen Partitionen (logical volumes). Diese können sich über mehrere Festplatten-Partitionen erstrecken und Funktionen wie Copy-on-Write bereitstellen.

Vorteile

- hohes Entwicklungsstadium
- gesicherte Weiterentwicklung
- unabhängig von Virtualisierungstechnik

Nachteile

- Aufwendige Einrichtung
- Live-Migration nicht möglich
- Nutzung von Sparse-Dateien schwer umsetzbar

2.6 Benchmarks

Ein wichtiger Punkt für die Entscheidung welche Copy-on-Write Implementierung optimal ist, ist die Lese- und Schreibgeschwindigkeit. Hierbei gibt es zwei Zugriffsarten, einmal den sequentiellen Zugriff und den wahlfreien oder auch zufälligen Zugriff.

2.6.1 Testbedingungen

Das Hostsystem für die Performance-Tests hat einen AMD Athlon II X2 250 Prozessor und 4 GiB RAM. Als Betriebssystem kommt sowohl auf Host- als auch Gastsystem ein 64 bit Debian squeeze zum Einsatz. Bei den KVM-Tests ist 2.6.32-5-amd64 der eingesetzte Kernel, für Xen wird der gleiche Kernel mit Xen-Unterstützung verwendet.

Während der Performance-Tests laufen neben der Virtuellen Maschine auf dem Hostsystem keine anderen aktiven Programme, die das Ergebnis verfälschen könnten. Als Referenz zu den Copy-on-Write-Techniken dient eine echte Festplattenpartition. Zum Testen der Performance werden IOzone und Bonnie++ eingesetzt.

IOzone

IOzone ist ein Tool mit dem in einer Reihe von unterschiedlichen Tests die Lese- und Schreib-Geschwindigkeit überprüft werden kann. Es wird hier zur Überprüfung der sequentiellen Lese- und Schreibgeschwindigkeit verwendet.

Bonnie++

Bonnie++ dient wie IOzone als Tool zum Testen von Festplatten. Es wird hier zur Überprüfung der sequentiellen Lese- und Schreibgeschwindigkeit sowie zum Testen des wahlfreien Zugriffs verwendet.

2.6.2 Testergebnisse

Die Testergebnisse werden in diesem Kapitel zusammenfassend aufgeführt und analysiert. Die kompletten Testergebnisse befinden sich im Anhang.

Die Abbildung 2.3 zeigt, dass mit KVM qcow2 gegenüber den anderen Copy-on-Write-Techniken einen Geschwindigkeitsvorteil beim sequentiellen Lesen und Schreiben hat. LVM-Snapshots und dm-snapshots liegen hingegen ungefähr gleich auf.

Abbildung 2.4 ist zu entnehmen, dass qcow2 wie auch bei den sequentiellen Tests vor LVM-Snapshots und dm-snapshots liegt. Der Unterschied zu der echten Festplattenpartition ist in beiden Tests sehr gering. Die guten Werte von qcow2 sowohl beim sequentiellen als auch beim zufälli-

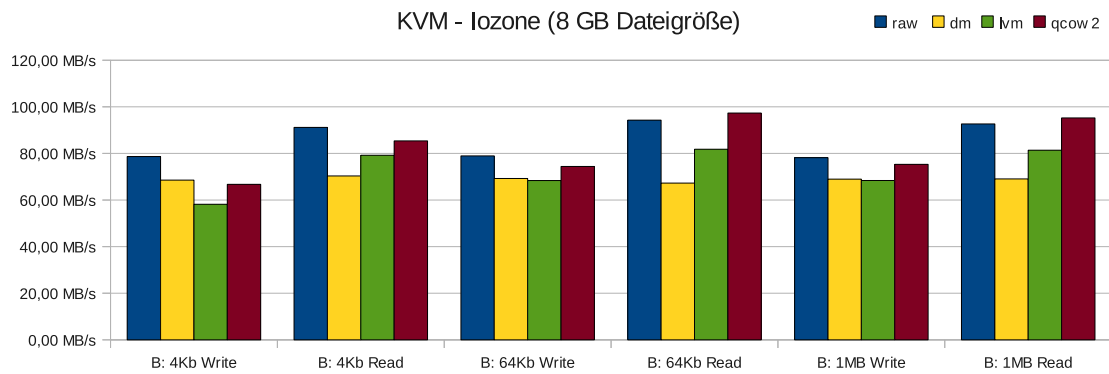


Abbildung 2.3: Iozone-kvm-8gb

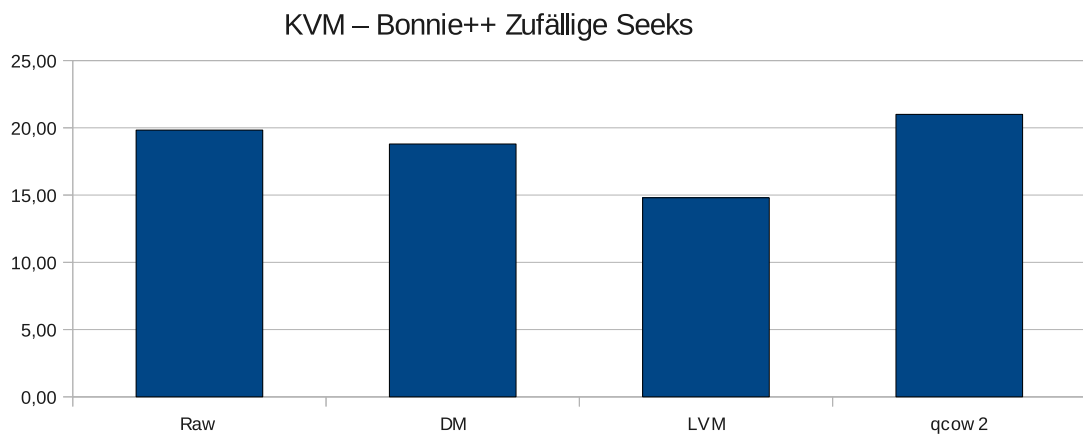


Abbildung 2.4: bonnie-kvm-random-seek

gem Zugriff auf die Festplatte, hängen mit der guten Integration in KVM zusammen.

In Xen schneiden die dm-snapshots besser ab als LVM-Snapshots und vhd beim sequentiellen Lesen und Schreiben, wie in Abbildung 2.5 zu sehen ist.

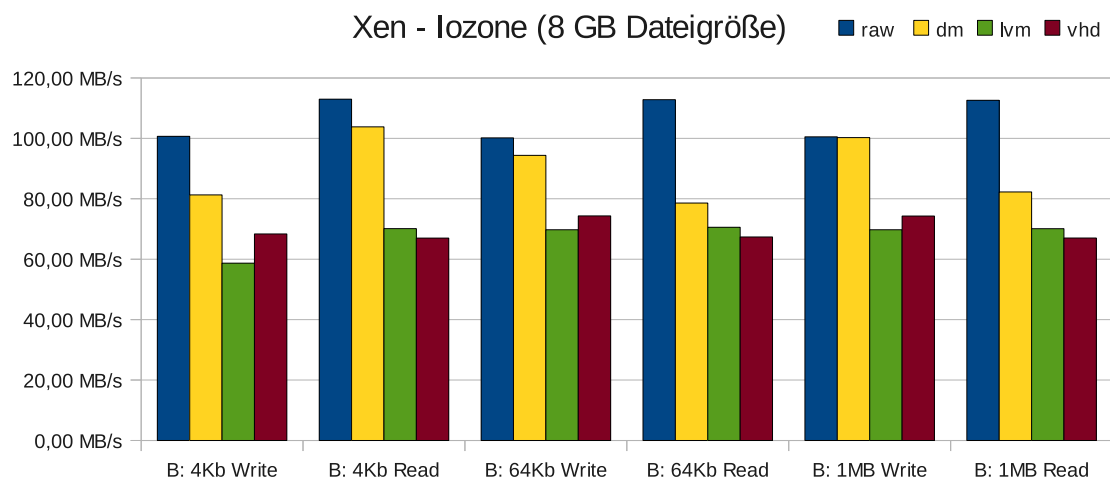


Abbildung 2.5: Iozone-xen-8gb

Beim zufälligen Zugriff auf die Festplatte ist unter Xen vhd abgeschlagen hinter LVM-Snapshots und dm-snapshots. Diese sind ungefähr gleichauf und liegen nicht weit hinter der Festplattenpartition (Abbildung 2.6). Das mittelmäßige Abschneiden des Imageformats vhd verwundert, da Citrix, die treibende Kraft der Xen Weiterentwicklung, eine optimierte vhd-Unterstützung entwickelt hat [Cro].

Die Testergebnisse zeigen, dass es Geschwindigkeitsunterschiede zwischen den Copy-on-Write-Techniken gibt. Diese Unterschiede in der Geschwindigkeit sind aber nicht so gravierend, dass man einzelne Copy-on-Write-Lösungen aufgrund der Performance-Tests kategorisch ausschließen muss-

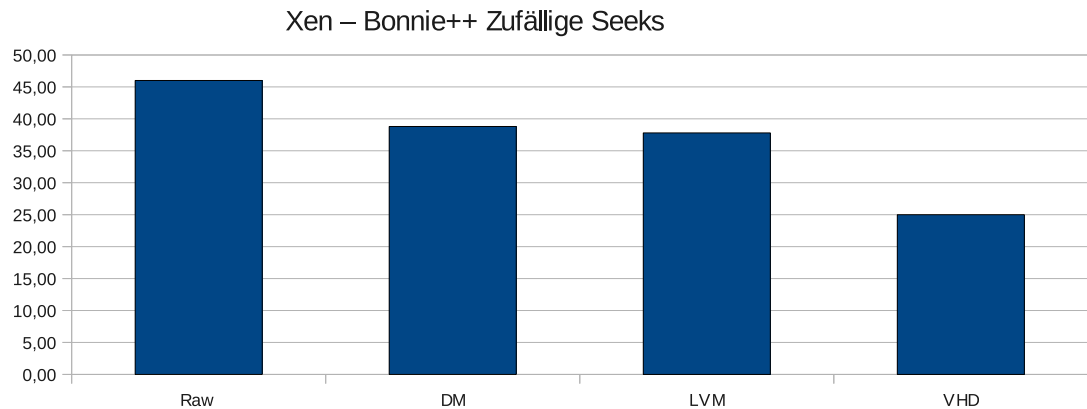


Abbildung 2.6: bonnie-xen-random-seek

te. Dennoch sind besonders die Vorteile von qcow2 in Verbindung mit KVM zu erwähnen. Für Xen gibt es kein Image-Format, das ähnliche Testergebnisse wie qcow2 in Verbindung mit KVM vorweisen kann.

2.7 Fazit

Es gibt bei den Testergebnissen keinen klaren Gewinner oder Verlierer. Im Großen und Ganzen fallen bei den Ergebnissen unter den einzelnen Copy-on-Write Verfahren keine bemerkenswerten Unterschiede auf. Aufgrund der unterschiedlichen Implementierungen der Copy-on-Write-Techniken in KVM und Xen, wird auch für die beiden Virtualisierungslösungen ein jeweiliges Fazit gezogen.

2.7.1 KVM

Unter KVM gibt es die Alternativen dm-snapshots LVM-Snapshots oder qcow2. Das von Microsoft entwickelte vhd kommt nicht in Frage. KVM unterstützt zwar das vhd-Format, jedoch nicht die Copy-on-Write-Funktion des Formats.

Die effizienteste Lösung für Copy-on-Write mit KVM ist qcow2. Dafür gibt es mehrere Gründe. Das qcow2-Format ist Teil des qemu-Projekts und damit sehr gut in dem darauf basierendem KVM integriert. Durch die gute Integration werden sehr gute Performance-Werte erreicht. Außerdem lässt es sich im Gegensatz zu dm-snapshots und LVM-Snapshots leichter administrieren.

2.7.2 Xen

Die für Xen zur Verfügung stehenden Copy-on-Write-Formate sind dm-snapshots, LVM-snapshots und VHD. Xen unterstützte in einigen vergangenen Versionen qcow2, diese Unterstützung ist jedoch nicht in der aktuellen Version enthalten (Version 4.0.1) [qco].

Für Xen ist VHD aktuell die attraktivste Copy-on-Write-Lösung. Es ist zwar laut der Performance-Tests nicht die schnellste Lösung, hat aber wesentliche Vorteile gegenüber dm-snapshots und LVM-Snapshots. Es werden keine Änderungen am Xen-Quelltext benötigt, wie es bei dm-snapshots der Fall ist [rac]. Die Funktion der Live-Migration ist mit vhd leichter zu realisieren als mit LVM-Snapshots und dm-snapshots. Die im weiteren Verlauf dieser Arbeit verwendete Lösung ist VHD. Falls Xen in den nächsten Versionen wieder qcow2 unterstützt, sollte jedoch die Verwendung von qcow2 auch unter Xen geprüft werden.

3 Analyse Verteilung von Images

Der Copy-on-Write-Mechanismus benötigt immer eine Vorlage - das Masterimage. Um es auf mehreren Virtualisierungsservern nutzen zu können, muss es über das Netzwerk verteilt werden oder über ein gemeinsam genutztes Storage-Backend zur Verfügung gestellt werden. Dieses Kapitel soll Wege aufzeigen diese Verteilung oder Bereitstellung möglichst effizient vorzunehmen.

Die Verteilungslösungen werden darauf überprüft, wie störanfällig sie sind. Ein anderer Punkt für die Entscheidungsfindung ist die benötigte Dauer der Verteilung. Außerdem wird einbezogen, wie skalierbar die Lösungen sind.

3.1 Multicast

Ein Multicast ist eine Mehrpunktverbindung. Der Sender schickt die Daten gleichzeitig an mehrere Empfänger. Durch das einmalige Senden an mehrere Empfänger wird Bandbreite eingespart. Die Daten werden nur an Rechner im Netz versendet diese auch angefordert haben, wie in Abbildung 3.1 schematisch dargestellt. Die Ausnahme bilden Switches die Multicasting nicht unterstützen, sie versenden die gesendeten Daten an

alle damit verbundenen Netzknoten [mul].

Da es bei den Masterimages darauf ankommt, dass sie komplett und fehlerfrei dupliziert werden, kann der Sender maximal so schnell senden, wie es der langsamste Empfänger entgegen nehmen kann. Dadurch ist die Verwendung von Multicast, in einer heterogenen Umgebung mit einem langsamen oder weit entfernten Empfänger, sehr ineffizient. Anwendung findet Multicast heute vor allem bei der Verteilung von Multimediadaten [Lei].

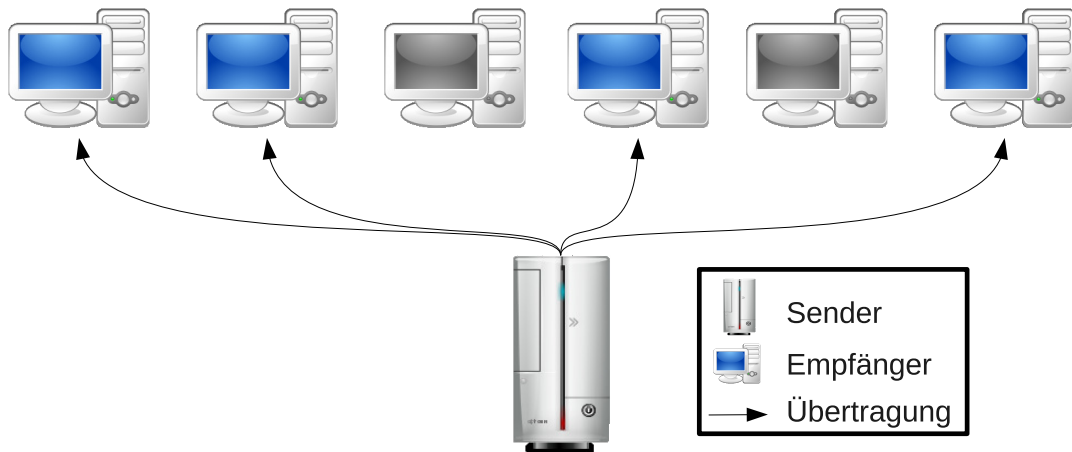


Abbildung 3.1: Multicast Beispiel

Vorteile

- sehr hohe Geschwindigkeit durch Parallelität

Nachteile

- hohe Netzwerklast
- Geschwindigkeitseinbruch bei heterogener Umgebung oder schlechten Netzanbindungen

3.2 BitTorrent

BitTorrent ist ein Netzwerkprotokoll zum effizienten Verteilen großer Dateien oder Sammlungen von Großen Dateien. Die Empfänger der Daten sind hierbei gleichzeitig auch Sender, sie werden Peers genannt [Coh08]. Damit wird nicht ein einziger zentraler Sender ausgelastet, sondern die Last wird auch auf alle Empfänger verteilt (zu sehen in Abbildung 3.2). Für die Kontaktaufnahme der Peers untereinander wird ein sogenannter Tracker benötigt. Aktuellere BitTorrent-Clients können aber auch trackerlos über eine verteilte Hashtabelle (engl. “Distributed Hash Table”; DHT) andere Peers finden [Loe08]. Durch den Einsatz von DHT kann die Einrichtung eines Trackers eingespart werden. Außerdem bringt es zusätzliche Ausfallsicherheit, da die Liste der verfügbaren Peers dezentral gespeichert wird.

Die zu übertragenden Daten werden nicht komplett in einem Stück übermittelt, sondern in Blöcke aufgeteilt. Bei zwischenzeitlichen Netzausfällen müssen somit auch nicht alle Daten noch einmal übertragen werden. Der BitTorrent-Client setzt nach dem Netzwerkausfall die Datenübertragung problemlos fort und muss nur gegebenenfalls die bereits übertragenen Daten einen Blockes verwerfen.

Vorteile

- hohe Skalierbarkeit
- Netzwerklast auf teilnehmende Netzwerksegmente beschränkt
- sehr effizient auch in heterogenen Umgebungen

Nachteile

- höherer Administrationsaufwand

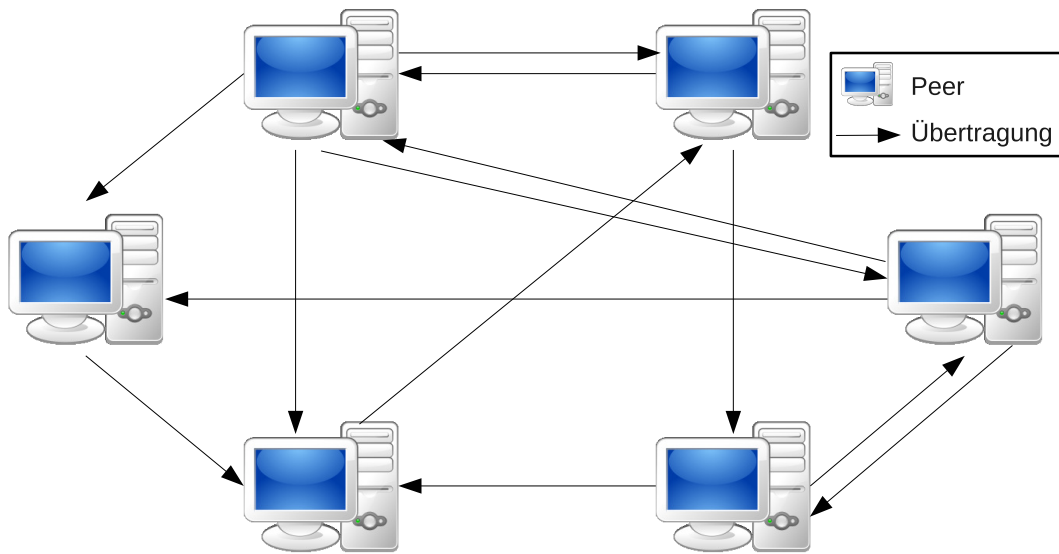


Abbildung 3.2: Bittorrent Beispiel

3.3 NFS

NFS (Network File System) ist ein Protokoll für das Bereitstellen von Daten über das Netzwerk. Das ist ein großer Unterschied zu den beiden vorher genannten Technologien. Die Daten werden nicht von einem Rechner auf den anderen kopiert, sondern über das Netzwerk wie eine lokale Festplatte zur Verfügung gestellt [nfs03]. Der Server macht hierbei eine Freigabe die von dem Clientrechner "gemountet" wird [nfs]. Die vom Clientrechner gemountete Freigabe wird in der Verzeichnisbaum eingebunden und kann wie lokales Verzeichnis angesteuert werden.

Vorteile

- geringer Administrationsaufwand

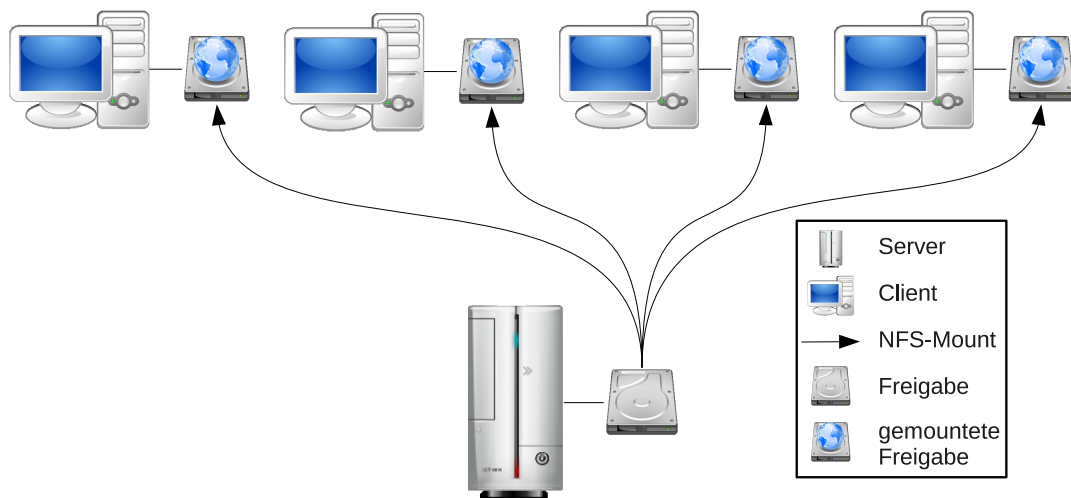


Abbildung 3.3: NFS Beispiel

Nachteile

- schlechte Skalierbarkeit
- viele von einer NFS-Freigabe gestartete virtuelle Maschinen, können zu einer permanent hohen Netzwerklast führen
- schlechte Lastenverteilung

3.4 Vergleich

Im Folgenden sollen die Verteilungsalternativen in Hinsicht auf die Kriterien Skalierbarkeit, Netzwerkausfall und Geschwindigkeit untersucht werden.

3.4.1 Skalierbarkeit

Eine gute Skalierbarkeit zeichnet sich dadurch aus, dass der Aufwand nicht signifikant ansteigt oder sich verlangsamt, wenn das Masterimage an einen weiteren Virtualisierungsserver verteilt wird. NFS zeigt dabei eine Schwäche, die Last steigt des NFS-Servers stetig mit jedem neuen NFS-Client an [Ker00].

Der Aufwand der Verteilung per Multicast steigt bei einem zusätzlichem Empfänger nicht an. Jedoch wird die Übertragung erheblich langsamer, wenn der zusätzliche Empfänger eine langsame Verbindung zu dem Server hat.

Der dezentrale Aufbau des BitTorrent-Netzes macht es sehr skalierbar. Jeder zusätzliche Empfänger des Masterimages, wird auch gleichzeitig zu einem Sender. Wenn der Upload und der Download gleich hoch sind, wird das Netz theoretisch also nicht langsamer. Das BitTorrent-Netz profitiert sogar von zusätzlichen Peers, da sie die Störanfälligkeit des Netzes verringern [EK].

3.4.2 Störanfälligkeit

Hier wird verglichen wie sich der Ausfall eines Netzwerkknotens auf die Verteilung auswirken. BitTorrent ist besonders unanfällig auf Ausfälle im Netz. Dieses wird durch die dezentrale Struktur ermöglicht. Wenn ein einzelner Netzwerkknoten ausfällt, besteht trotzdem unter den noch verfügbaren Knoten ein Netz.

NFS und Multicast haben im Unterschied zu BitTorrent einen wunden Punkt, da die Verteilung über einen einzigen Knoten stattfindet. Der Ausfall eines bestimmten Knotens führt also zum kompletten Ab-

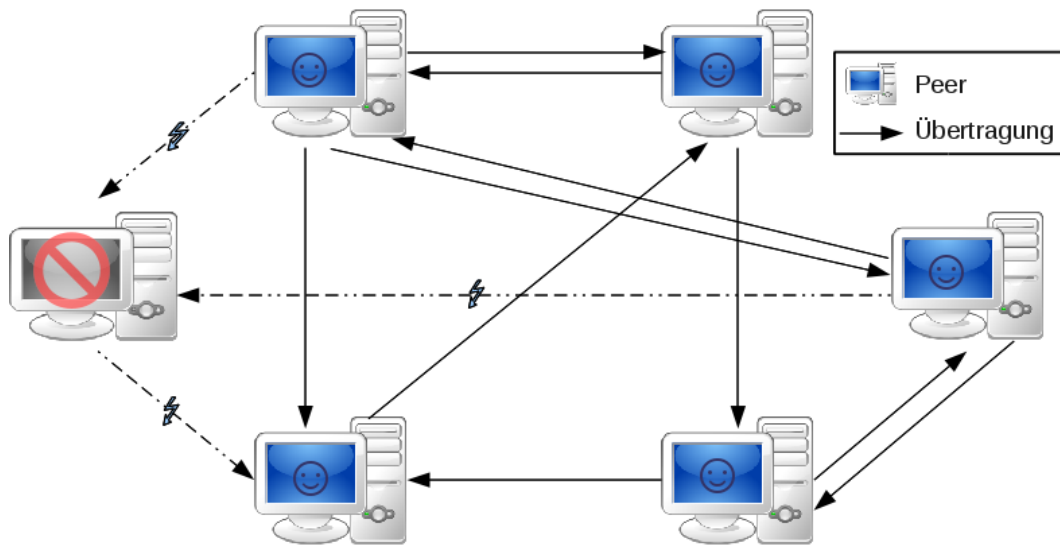


Abbildung 3.4: Bittorrent Netzwerkausfall

bruch der Verteilung. Man nennt diesen Punkt *Single Point of Failure*.

Bei NFS gibt es beim Bereitstellen der Masterimages zusätzlich die Problematik, dass der Festplattenzugriff der virtuellen Maschinen von der Verfügbarkeit des NFS-Servers abhängt. Ein Ausfall führt damit zu dem Abstürzen der virtuellen Maschinen.

3.4.3 Verteilungsdauer

Besonders hervorzuheben ist NFS, da es nicht wie BitTorrent und Multicast die Masterimages verteilt sondern bereitstellt. Dadurch benötigt es keine Zeit die Masterimages zu verteilen und kann sie direkt zur Verfügung stellen.

Die Dauer der Übertragung ist bei Multicast vom langsamsten betei-

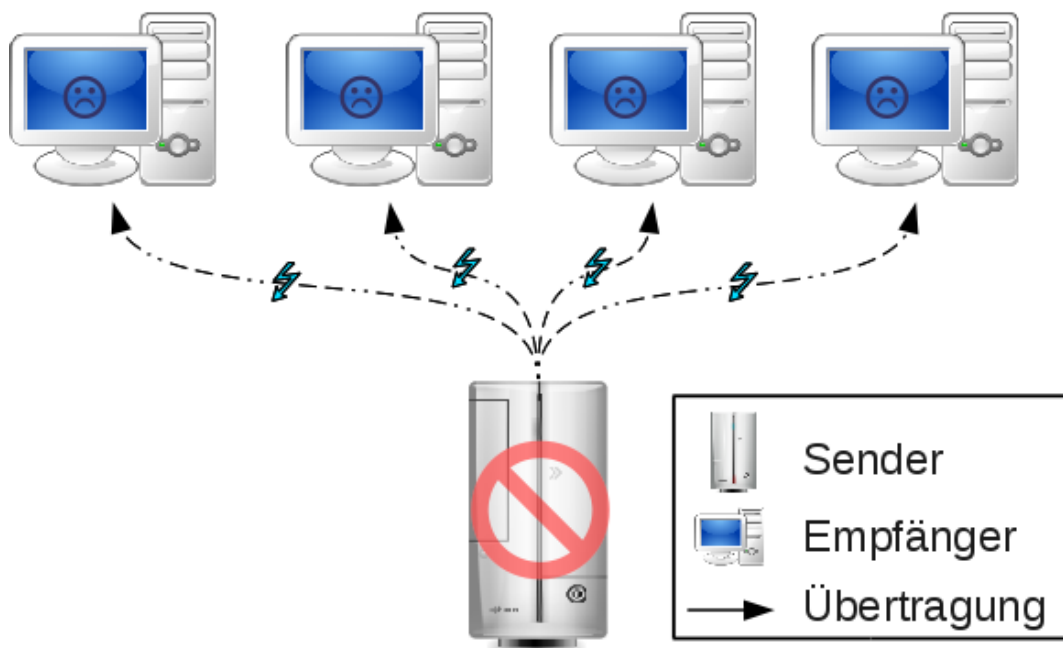


Abbildung 3.5: Multicast Netzwerkausfall

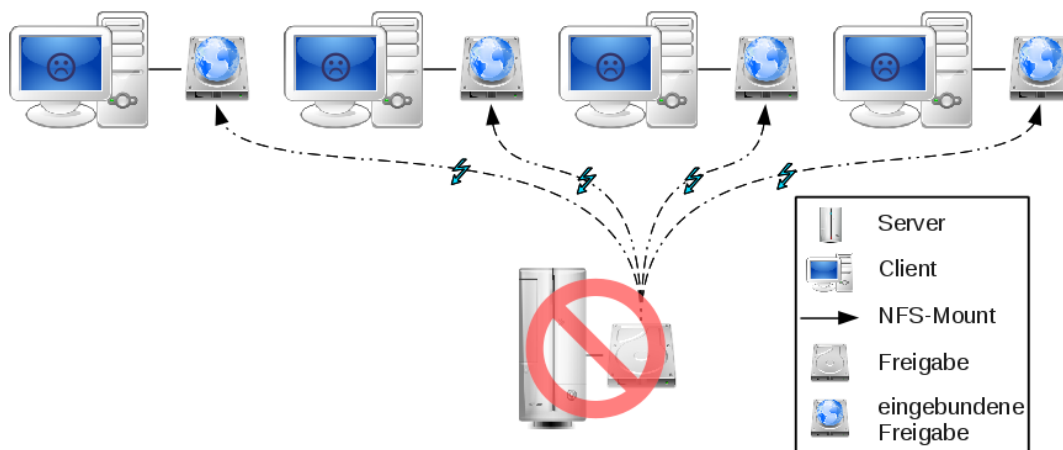


Abbildung 3.6: NFS Netzwerkausfall

ligten Netzwerkknoten abhängig. Ideal ist es, wenn alle Empfänger und der Sender über die gleiche Download- und Upload-Bandbreite verfügen (homogene Umgebung). So kann die gleichzeitige Übertragung an alle Empfänger optimal ausgenutzt werden.

BitTorrent zeichnet sich vor allem durch aus, dass es auch gute Ergebnisse erzielt, wenn die Peers über unterschiedliche Download- und Upload-Geschwindigkeiten verfügen. In einer homogenen Umgebung benötigt es mehr Zeit für die Verteilung als Multicast.

3.5 Fazit

Alle aufgezeigten Lösungen für das Verteilen von Masterimages haben ihre Vor- und Nachteile. Jedoch zeigt sich, dass BitTorrent wesentliche Vorteile gegenüber den anderen beiden Lösungen hat. Eine geringe Störanfälligkeit ist im produktiven Einsatz sehr wichtig. Auf diesem Gebiet liegt BitTorrent weit vor NFS und Multicast. Multicast ist ein sehr effizientes Protokoll und es wird auf produktiv eingesetzt, zum Beispiel bei der Übertragung von Multimediadaten. Allerdings ist es ungeeignet wenn der Empfänger alle Daten erhalten muss. Auch die Erweiterbarkeit um zusätzliche Virtualisierungsserver unterstützt die Schlussfolgerung, dass BitTorrent für den hier diskutierten Einsatz die effizienteste Lösung ist.

4 Synthese

Die Erkenntnisse des vorhegenden Kapitels werden in diesem Kapitel aufgegriffen und zu ein Konzept für Komplettlösung zusammengefügt. Dieses Konzept wird im Anschluss auf dem geeignetem Wege umgesetzt.

4.1 Konzept

Zweck der entwickelten Verwaltungslösung ist es nicht alle Aspekte Virtualisierung zu verwalten. Sie dient dem Zweck virtuelle Maschinen sehr schnell zu replizieren und die dafür benötigten Vorlagen schnell auf den Virtualisierungsservern zu verteilen. Die einzelnen Vorraussetzungen die dafür nötig sind, werden hier dargelegt.

4.1.1 Steuerung und Kommunikation

Um Masterimages in einem Netz mit mehreren Virtualisierungsservern zu verteilen und zu klonen, bedarf es einer Kommunikation zwischen den Rechnern. Diese Kommunikation sollte von einem zentralen Server gesteuert werden. Dieser Verwaltungsserver kann selbst ein Virtualisierungsserver sein oder ausschließlich mit der Verwaltung beschäftigt sein.

Die Virtualisierungstechniken sollen über eine einheitliche Schnittstelle verwaltet werden. Durch die einheitliche Schnittstelle wird die Verwaltung vereinfacht und zusätzlicher Aufwand vermieden. Das Starten Stoppen und das Definieren virtueller Maschinen erfolgt über zentrale Schnittstelle. Auch die Möglichkeit bei einer Weiterentwicklung des Programms neue Virtualisierungstechniken zu integrieren soll gegeben sein.

4.1.2 Verteilung

Die Verteilung der Masterimages findet über das BitTorrent-Protokoll statt (siehe Kapitel 3). Der BitTorrent-Client muss für eine einfache und automatisierte Verteilung über die Kommandozeile bedienbar sein. Eine weitere Voraussetzung ist die Unterstützung des Protokolls DHT. DHT ermöglicht das Finden anderer Peers ohne zentralen Tracker. Das ermöglicht es ein BitTorrent-Netzwerk aufzubauen ohne einen Tracker einrichten zu müssen.

Zum Starten der Verteilung der Masterimages wird zunächst eine .torrent-Datei erstellt und an alle Virtualisierungsserver gesendet, die es erhalten sollen. Danach wird der BitTorrent-Client gestartet und der Download initiiert.

Nicht jeder Virtualisierungsserver kann das Verteilen initiieren, sondern nur das Verwaltungsprogramm des Verwaltungsservers. Dies gewährleistet, dass nicht jeder Virtualisierungsserver auf jeden anderen zugreifen kann.

4.1.3 Klonen

Das Klonen wird, wie auch die Verteilung, von dem zentralen Verwaltungsserver verwaltet. Für das eigentliche Klonen der virtuellen Festplatten werden die in den Virtualisierungstechniken integrierten Programme eingesetzt.

4.2 Realisierung einer Komplettlösung

In diesem Unterkapitel werden die oben genannten Punkte aufgegriffen und deren Umsetzung beschrieben.

4.2.1 Rahmenbedingungen

Die Komplettlösung wird auf einem Debian squeeze System implementiert. In der Komplettlösung werden ein paar wenige debianspezifische Befehle wie zum Beispiel *apt-get* verwendet. Diese können aber leicht für andere Linux-Distributionen portiert werden. Neben den oben genannten Software kommen ssh und rsync zum Einsatz.

Für die Programmierung wird die Skriptsprache Python eingesetzt. Da das hier entwickelte Verwaltungsprogramm nicht zeitkritisch ist, hat die Performanz keine hohe Priorität. Viel wichtiger ist es den Wartungsaufwand niedrig zu halten. Mit diesen Bedingungen ist die Skriptsprache Python eine sehr gute Wahl.

4.2.2 Steuerung und Kommunikation

Um die Steuerung der Virtualisierungsserver zu vereinfachen und zu vereinheitlichen wird in dieser Arbeit die Virtualisierungs-API libvirt verwendet. Die Virtualisierungstechniken Xen und KVM können beide mit libvirt verwaltet werden. Die Fähigkeiten von libvirt umfassen zum Beispiel das Erstellen, Starten, Stoppen, Pausieren sowie die Migration von virtuellen Maschinen.

Alle virtuellen Maschinen werden von libvirt als XML-Beschreibung verwaltet. Sie enthalten Informationen zu der virtuellen Hardware und eine eindeutige Identifikationsnummer. Eine solche XML-Beschreibung ist beispielhaft im Folgenden zu dargestellt.

```
1 <domain type='kvm'>
2   <name>debian</name>
3   <memory>512000</memory>
4   <currentMemory>512000</currentMemory>
5   <vcpu>1</vcpu>
6   <os>
7     <type>hvm</type>
8     <boot dev='hd' />
9   </os>
10  <features>
11    <acpi />
12  </features>
13  <clock offset='utc' />
14  <on_poweroff>destroy</on_poweroff>
15  <on_crash>destroy</on_crash>
16  <devices>
17    <emulator>/usr/bin/kvm</emulator>
18    <disk type='file' device='disk'>
19      <driver name='qemu' type='qcow2' />
20      <source file='/var/lib/libvirt/images/debian.qcow2' />
21      <target dev='hda' />
22    </disk>
23    <interface type='network'>
24      <source network='default' />
25    </interface>
```

```

26     <input type='mouse' bus='ps2' />
27     <graphics type='vnc' port='-1' listen='0.0.0.0' />
28 </devices>
29 </domain>

```

Listing 4.1: libvirt-XML Beispiel

Libvirt bietet die Möglichkeit über das Netzwerk angesprochen zu werden. Außerdem unterstützt libvirt neben Xen und KVM noch andere Virtualisierungstechniken, die bei einer weiteren Entwicklung in die Komplettlösung integriert werden können.

Die einzelnen Aufgaben wie das Verteilen und das Klonen werden auf den Virtualisierungsservern von lokal installierten Skripten erledigt. So kann vermieden werden, dass unnötig viele Befehle über das Netzwerk gesendet werden müssen. Die Skripte werden über das Netzwerkprotokoll ssh gestartet.

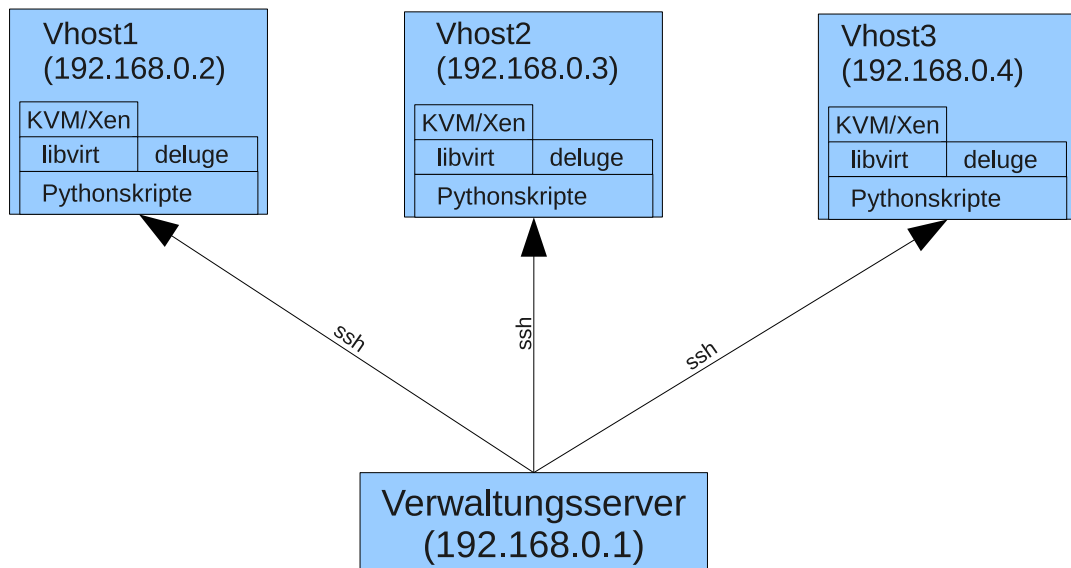


Abbildung 4.1: Kommunikation

4.2.3 Einrichtung eines Virtualisierungshosts

Die Einrichtung wird durchgeführt um auf allen verwalteten Virtualisierungshosts die Grundvoraussetzungen für das Klonen und Verteilen zu schaffen. Während der Einrichtung wird die nötige Software installiert und es werden Einstellungen vorgenommen. Sie ermöglichen das einfache Kopieren und Klonen von virtuellen Maschinen. Der Ablauf der Einrichtung wird im Folgenden dargelegt.

Ablauf

Der Benutzer gibt zunächst die Hostadresse des Virtualisierungshosts an. Ebenfalls wird die Virtualisierungstechnik des neuen Hosts abgefragt. Nach der Eingabe wird der Rechnername abgerufen. Für die einfache Kommunikation wird auf Host der ssh-key des Verwaltungsservers hinzugefügt.

Um nicht alle Aktionen remote über das Netzwerk ausführen zu müssen, werden die Funktion des Klonens und der Verteilung in Skripte ausgelagert. Diese Skripte werden von dem Verwaltungsserver auf den neuen Host übertragen.

Im Anschluss folgt die Installation der benötigten Software-Pakete. Es werden die Pakete für libvirt, deluge, sowie für administrative Tools installiert.

Für die Abfrage über das Netzwerk verwendet libvirt Zertifikate. Es gibt drei unterschiedliche Zertifikate. Das Server-Zertifikat dient dazu die Echtheit des Virtualisierungshosts zu validieren. Das Client-Zertifikat wird von dem Server dazu verwendet, den Client zu validieren und ihm somit Zugriff zu gewähren. Das CA-Zertifikat wird benötigt um die an-

deren beiden Zertifikate zu erstellen und zu zertifizieren. Das Server-Zertifikat wird von dem Verwaltungsserver erstellt. Nach der Erstellung wird es auch dem Virtualisierungsserver abgelegt. (**Hinweis:** Dies ist nur eine vereinfachte Darstellung des Zertifikate-Systems. Eine ausführliche Beschreibung ist unter <http://wiki.libvirt.org/page/TLSSetup> zu finden.)

Die verwalteten Virtualisierungsserver werden in als Liste in einer Klartext Datei abgespeichert. Sie enthält zu jedem Virtualisierungsserver den Rechnernamen sowie die Virtualisierungstechnik.

4.2.4 Verteilung

Für den Zweck der Verteilung, kommt in dieser Arbeit *deluge* als BitTorrent-Client zum Einsatz. Er kann komplett über die Kommandozeile gesteuert werden und hat die Möglichkeit per DHT andere Peers zu finden.

Ablauf

Zunächst wählt der Benutzer einen Virtualisierungs-Host aus, der die zu verteilende virtuelle Maschine beherbergt.

```
1 def chooseVHost():
2     hList = hostList()
3     if hList:
4         print 'Wählen Sie den Virtualisierungshost aus:'
5         print 'ID\tHost\tTyp'
6         for host in hList:
7             print str(host[0]) + '\t' + host[1] + '\t' + host[2]
8             hostId = intInput('ID:')
9             return [hList[hostId][1], hList[hostId][2]]
10    else:
11        print 'kein Virtualisierungshost vorhanden'
12        return [None, None]
```

Listing 4.2: VHost-Auswahl

Die Methode für die Auswahl lässt den Benutzer zwischen allen ausgeschalteten virtuellen Maschinen auswählen.

```
1 def chooseVm(hostName, vType):
2     vOfffList = vmOfffList(hostName, vType)
3     if vOfffList:
4         print 'Wählen Sie eine VM aus:'
5         print 'ID\tName\tState'
6         for i in range(0, len(vOfffList)):
7             print str(i) + '\t' + vOfffList[i].name() + '\t' + str(
                vOfffList[i].info()[0])
8     vmId = intInput('ID: ')
9     return vOfffList[vmId]
```

Listing 4.3: VM-Auswahl

Außerdem gibt der Benutzer an welche Virtualisierungsserver die virtuelle Maschine verteilt werden soll. Nach der Auswahl der Server und der VM, erstellt das Skript `maketorrent.py` (siehe 4.2.3) eine torrent-Datei aus der XML-Beschreibung von libvirt und den virtuellen Festplatten. Sie wird an alle ausgewählten V-Hosts mit `rsync` weitergegeben. Nun werden alle BitTorrent-Clients gestartet und die erstellte torrent-Datei hinzugefügt.

4.2.5 Klonen

Für das Klonen der virtuellen Maschinen werden die von den Virtualisierungstechniken mitgebrachten Tools verwendet. Auf einem Xen-Server ist es das Tool *vhd-util*, bei KVM *kvm-img*. Um die virtuelle Maschine zu klonen müssen Änderungen an der XML-Beschreibung vorgenommen werden und die Festplatten mit den Tools der Virtualisierungstechniken

von der Vorlage abgeleitet werden. Der Ablauf des Klonens wird im Folgenden beschrieben.

Ablauf

Beim Klonen einer virtuellen Maschine wählt der Benutzer, wie bei der Verteilung, einen Virtualisierungs-Host und eine virtuelle Maschine aus (siehe 4.3). Zusätzlich dazu wird die Anzahl der Klone und die Option alle Klone sofort zu starten abgefragt. Nach den erfolgten Benutzereingaben ruft das Programm das auf dem Virtualisierungsserver befindliche Skript `clone.py` zum Klonen auf.

Im ersten Schritt des Klonvorgangs generiert das Skript einen neuen Namen. Der Name setzt sich aus dem alten Namen und 6 zufälligen und Buchstaben zusammen. Der nächste Schritt ist es die Beschreibung der Vorlage aus libvirt zu laden. Aus ihr werden die Festplatten der Vorlage ausgelesen und geklont. Die Identifikationsnummer und die MAC-Adresse aus der Beschreibung werden gelöscht und der neue Name eingetragen. Die MAC-Adresse und die Identifikationsnummer generiert libvirt neu beim Anlegen der geklonten VM. Die Änderungen an der XML-Beschreibung sind in dem Listing 4.4 zu sehen. Alle entfernten Zeilen sind rot markiert, alle hinzugefügten grün.

```
1  <domain type='kvm'>
2  -  <name>debian</name>
3  -  <uuid>a6a02d47-6255-7ca7-79e7-22b2cde046a7</uuid>
4  +  <name>debianVxyIZ5</name>
5  +
6      <memory>512000</memory>
7      <currentMemory>512000</currentMemory>
8      <vcpu>1</vcpu>
9      .
10     .
11     .
12  <devices>
13     <emulator>/usr/bin/kvm</emulator>
```

```

14     <disk type='file' device='disk'>
15         <driver name='qemu' type='qcow2' />
16 -         <source file='/var/lib/libvirt/images/debian.qcow2' />
17 +         <source file='/var/lib/libvirt/images/debianVxyIZ5-debian.
qcow2' />
18         <target dev='hda' />
19     </disk>
20     <interface type='network'>
21 -         <mac address='52:54:00:3d:eb:4b' />
22 +
23         <source network='default' />
24     </interface>
25     <input type='mouse' bus='ps2' />
26     <graphics type='vnc' port='-1' listen='0.0.0.0' />
27 </devices>

```

Listing 4.4: modifizierte XML-Beschreibung

4.2.6 Skripte/Programme

Die einzelnen Aufgaben der Verwaltungslösung werden nicht in einem einzigen Skript abgearbeitet, sondern in einzelne übersichtliche Skripte aufgeteilt. In diesem Unterkapitel werden sie aufgelistet und ihre Funktion beschrieben.

Verwaltungsserver

Die Skripte des Verwaltungsserver erledigen die Verwaltungs und Einrichtungsaufgaben wahr. Ihre Aufgaben werden hier kurz erklärt.

cow.py - Hauptprogramm der Verwaltungslösung es nimmt die Benutzereingaben entgegen und ruft die anderen Skripte auf.

hostname.sh - fragt den Rechnernamen des Virtualisierungsserver ab und trägt dort den ssh-key des Verwaltungsservers ein

servercert.sh - erstellt ein Server-Zertifikat für den Virtualisierungsserver

cacert.sh - erstellt das CA-Zertifikat wenn es nicht vorhanden ist und überträgt den öffentlichen Schlüssel des CA-Zertifikats an den Virtualisierungsserver

clientcert.sh - erstellt ein Client-Zertifikat, dass dem Verwaltungsserver Zugriff auf die libvirt-Installationen der Virtualisierungsserver ermöglicht

Virtualisierungsserver

Diese Skripte liegen auf dem Virtualisierungsserver, sie werden während der Einrichtung vom Verwaltungsserver installiert (siehe 4.2.3) , um lokal abrufbar zu sein. Ihre Funktionen werden im Folgenden kurz erläutert.

clone.py - Klonen der virtuellen Maschinen, es nimmt die nötigen Modifikationen an der XML-Beschreibung der Vorlage vor und klonet die virtuellen Festplatten

maketorrent.py - erzeugt eine torrent-Datei und startet das Verteilen mit dem BitTorrent-Client

packageinstall.py - installiert die benötigten Software-Pakete auf dem Virtualisierungsserver

whoami.py - legt eine Konfigurationsdatei mit Informationen zu dem Virtualisierungsserver

xenprep.py - nimmt Einstellungen an dem Xen-Daemon vor, damit libvirt den Xen-Daemon abfragen kann

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

5.2 Fazit

5.3 Ausblick

6 Anhang

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  import os
5  import re
6  import subprocess
7  import shutil
8  import sys
9  import libvirt
10 import socket
11 import time
12 from xml.dom.minidom import parseString
13
14 vHostType = {1: 'kvm', 2: 'xen'}
15 downloadDir = '/var/lib/download'
16 imageDir = '/var/lib/libvirt/images'
17 binDir = '/opt/cow'
18
19 def debugOut(output, debugLevel):
20     if debug >= debugLevel:
21         print 'Debug_[' + str(debugLevel) + ']: ' + output
22
23 def hostList():
24     hostList = []
25     hosts = open(os.path.expanduser('~/.cow/vhosts'), 'r').
26         readlines()
27     for i in range(0, len(hosts)):
28         host = hosts[i].split('\t')
29         if len(host) == 2: #ignore malformed rows
30             hostList.append([len(hostList), host[0], host[1].replace('\n', '')])
31     return hostList
32
33 def vmList(hostName, vType):
```



```

33     try:
34         vmList = vmOffList(hostName, vType) + vmOnList(hostName,
35             vType)
36         return vmList
37     except libvirt.libvirtError:
38         print 'Host_' + hostName + '(' + vType + ') nicht erreichbar'
39
40 def vmOffList(hostName, vType):
41     vOffList = []
42     if vType == 'xen':
43         conn = libvirt.open('xen://' + hostName + '/')
44     else:
45         conn = libvirt.open('qemu://' + hostName + '/system')
46
47     for name in conn.listDefinedDomains():
48         vOffList.append(conn.lookupByName(name))
49
50     return vOffList
51
52 def vmOnList(hostName, vType):
53     vOnList = []
54     if vType == 'xen':
55         conn = libvirt.open('xen://' + hostName + '/')
56     else:
57         conn = libvirt.open('qemu://' + hostName + '/system')
58
59     for id in conn.listDomainsID():
60         vOnList.append(conn.lookupByID(id))
61
62     return vOnList
63
64 def vHostExists(hostName):
65     hList = hostList()
66     for host in hList:
67         if host[1] == hostName:
68             return True
69     return False
70
71 def newVServer():
72     print 'Geben Sie die IP des Hosts ein'
73     ip = raw_input('HostIP: ')
74
75     print 'Virtualisierungstechnik'
76     print '1: KVM'

```

```

77     print '\u2: \uXEN'
78     choice = intInput('')
79
80     command = ['./hostname.sh', ip]
81     hostName, stderr = execute(command)
82     hostName = hostName.replace('\n', '')
83
84     if vHostExists(hostName):
85         print 'der \uHost \u' + hostName + ' \uist \uschon \uregistriert'
86     else :
87         #append the new host
88         vhosts = open(os.path.expanduser('~/.cow/vhosts'), 'a')
89         vhosts.write('\n' + hostName + '\t' + vHostType[choice])
90         vhosts.close()
91
92         command = ['rsync', '-r', 'client-scripts/' , 'root@' +
93                   hostName + ':' + binDir]
94         execute(command)
95
96         command = ['ssh', 'root@' + hostName, '/opt/cow/
97                   packageinstall.py \u"deluged \u deluge-console \u mktorrent \u
98                   libvirt-bin \u python-libvirt \u dnsmasq-base" ' ]
99         execute(command)
100
101         command = ['./cacert.sh', hostName]
102         execute(command)
103
104         #command = ['./clientcert.sh', hostName]
105         #execute(command)
106
107         command = ['ssh', 'root@' + hostName, 'mkdir \u-p \u' + binDir ]
108         execute(command)
109
110         command = ['ssh', 'root@' + hostName, binDir + '/whoami.py \u'
111                   + ip + ' \u' + downloadDir + ' \u' + imageDir ]
112         execute(command)
113
114 def clone():
115     hostName, vType = chooseVHost()
116     if hostName:
117         try:
118             vm = chooseVm(hostName, vType)

```

```

118         if vm:
119             cloneCount = raw_input('Anzahl der Klone: ')
120             autostart = raw_input('Nach Erstellung starten? (j/n): ')
121             command = ['ssh', 'root@' + hostName, binDir + '/clone.py',
                        ' ' + vm.name() + ' ' + cloneCount + ' ' + autostart +
                        ' ' + str(debug)]
122             stdout, stderr = execute(command)
123             print stdout
124         else:
125             print 'keine passende VM vorhanden'
126     except libvirt.libvirtError:
127         print 'Host ' + hostName + '(' + vType + ') nicht erreichbar'
128
129 def overview():
130     hList = hostList()
131     if hList:
132         for host in hList:
133             vList = vmList(host[1], host[2])
134             if vList:
135                 print '\nHost: ' + host[1] + ' (' + host[2] + ')\n'
136                 print 'ID\tName'
137                 for vm in vList:
138                     print vm.name() + '\t' + str(vm.info()[0])
139
140 def initialize():
141     if not os.path.exists(os.path.expanduser('~/.cow/')):
142         os.mkdir(os.path.expanduser('~/.cow/'))
143
144     if not os.path.exists(os.path.expanduser('~/.cow/vhosts')):
145         open(os.path.expanduser('~/.cow/vhosts'), 'w').close()
146
147     if not os.path.exists('/var/log/cow.log'):
148         open('/var/log/cow.log', 'w').close()
149
150 def execute(command):
151     s = subprocess.Popen(command, stdout=subprocess.PIPE, stdin=
        subprocess.PIPE, stderr=subprocess.PIPE)
152     stdout, stderr = s.communicate()
153     debugOut(stdout, 3)
154     debugOut(stderr, 1)
155     return stdout, stderr
156
157 def hddList(xmlDescription):
158     hList = []

```

```

159 description = parseString(xmlDescription)
160 hardDisks = description.getElementsByTagName('disk')
161 for i in range(0,len(hardDisks)):
162     if hardDisks[i].getAttribute('device') == 'disk':
163         hardDisk = hardDisks[i].getElementsByTagName('source')[0].
            getAttribute('file')
164     hList.append(hardDisk)
165 return hList
166
167 def chooseVHost():
168     hList = hostList()
169     if hList:
170         print 'Wählen_Sie_den_Virtualisierungshost_aus:'
171         print 'ID\tHost\tTyp'
172         for host in hList:
173             print str(host[0]) + '\t' + host[1] + '\t' + host[2]
174         hostId = intInput('ID: ')
175         return [hList[hostId][1], hList[hostId][2]]
176     else:
177         print 'kein_Virtualisierungshost_vorhanden'
178         return [None, None]
179
180 def chooseVHosts(filterName, filterVType):
181     print 'Verteilen_an_Hosts:'
182     hList = hostList()
183     for host in hList:
184         if host[2] == filterVType:
185             print str(host[0]) + '\t' + host[1] + '\t' + host[2]
186         targetHostIds = raw_input('IDs_(z.B_0,2,3): ')
187         #remove whitespaces and split the comma seperated value
188         targetHostIds = targetHostIds.replace('\t', '').replace(' ', '')
            ).split(',')
189     #remove duplicates
190     targetHostIds = list(set(targetHostIds))
191     targetHosts = []
192     for i in targetHostIds:
193         if hList[int(i)][1] != filterName:
194             targetHosts.append(hList[int(i)])
195     return targetHosts
196
197 def chooseVm(hostName, vType):
198     vOffList = vmOffList(hostName, vType)
199     if vOffList:
200         print 'Wählen_Sie_eine_VM_aus:'
201         print 'ID\tName\tState'

```

```

202     for i in range(0,len(vOffList)):
203         print str(i) + '\t' + vOffList[i].name() + '\t' + str(
                vOffList[i].info()[0])
204     vmId = intInput('ID: ')
205     return vOffList[vmId]
206
207 def startDownload(vHostList, vmName):
208     debugOut('start_the_deluge_daemons_on_target_hosts', 1)
209     #start the deluge daemon
210     for targetHost in vHostList:
211         debugOut('starting_deluge_daemon_on' + targetHost[1], 1)
212         command = ['ssh', 'root@' + targetHost[1], 'deluged']
213         s = subprocess.Popen(command)
214     #waiting for the deluge daemon
215     time.sleep(1)
216     for targetHost in vHostList:
217         debugOut('rsync_to' + targetHost[1], 2)
218         command = ('rsync', '/tmp/' + vmName + '.torrent', 'root@' +
                targetHost[1] + ':' + downloadDir + '/' + vmName + '.
                torrent')
219         debugOut(repr(command),3)
220         execute(command)
221         command = ['rsync', '/tmp/' + vmName + '.xml', 'root@' +
                targetHost[1] + ':/tmp/']
222         execute(command)
223         command = ['ssh', 'root@' + targetHost[1], 'virsh_define_/
                tmp/' + vmName + '.xml']
224         execute(command)
225
226         debugOut('del_torrent_on' + targetHost[1], 3)
227         command = ['ssh', 'root@' + targetHost[1], 'deluge-console'
                'del' + vmName + '"]
228         s = subprocess.Popen(command)
229     #waiting for deletion of the old torrent
230     time.sleep(0.5)
231     #add the torrent and start downloading
232     for targetHost in vHostList:
233         debugOut('start_torrent_on' + targetHost[1],3)
234         command = ['ssh', 'root@' + targetHost[1], 'deluge-console'
                'add' + downloadDir + '/' + vmName + '.torrent"]
235         s = subprocess.Popen(command)
236
237
238 def shareImage():
239     hostName, vType = chooseVHost()

```

```

240 if hostName:
241     try:
242         vm = chooseVm(hostName, vType)
243         if vm:
244             targetHosts = chooseVHosts(hostName, vType)
245             targetHosts.append([-1, hostName, vType])
246             command = ['ssh', 'root@' + hostName, '/opt/cow/
                maketorrent.py' + vm.name()]
247             stdout, stderr = execute(command)
248             if stderr:
249                 print "Fehler:" + stderr
250                 return
251             command = ['rsync', 'root@' + hostName + ':' +
                downloadDir + '/' + vm.name() + '.torrent', '/tmp/']
252             execute(command)
253             command = ['rsync', 'root@' + hostName + ':' +
                downloadDir + '/' + vm.name() + '/' + vm.name() + '.
                xml', '/tmp/']
254             execute(command)
255
256             startDownload(targetHosts, vm.name())
257         else:
258             print 'keine VM vorhanden'
259     except libvirt.libvirtError:
260         print 'Host' + hostName + '(' + vType + ') nicht
            erreichbar'
261
262 def test():
263     print 'test'
264
265 def intInput(output):
266     tmpString = raw_input(output)
267     try:
268         intNum = int(tmpString)
269     except ValueError:
270         print 'Keine gültige Zahl'
271         intNum = intInput(output)
272     return intNum
273
274 options = {
275     1 : newVServer,
276     2 : shareImage,
277     3 : clone,
278     4 : overview,
279     0 : test}

```

```

280
281 initialize()
282
283 #set debugLevel
284 if len(sys.argv) > 1:
285     try:
286         debug = sys.argv[1]
287     except ValueError:
288         debug = 0
289 else:
290     debug = 0
291
292 while (True):
293     print 'Menü'
294     print '1: neuer Virtualisierungsserver'
295     print '2: Image verteilen'
296     print '3: Virtuelle Maschine klonen'
297     print '4: Übersicht aller virtuellen Maschinen'
298     print '5: Beenden'
299
300     choice = intInput('Auswahl: ')
301
302     if choice <= 4 and choice >= 0:
303         options[choice]()
304     else:
305         sys.exit(0)

```

Listing 6.1: cow.py

```

1  #!/bin/bash
2  ip="$1"
3  host=$(ssh root@$ip 'echo "$HOSTNAME"')
4  ssh-copy-id root@$host > /dev/null
5
6  if [ ! $(grep -l "$host[ ]*$" "/etc/hosts") ]
7  then
8      echo -e "$ip\t$host" >> /etc/hosts
9  fi
10
11  echo $host
12  exit 0

```

Listing 6.2: hostname.sh

```

1  #!/bin/bash
2  host=$1

```

```

3 if [ ! -e /etc/pki/CA/cacert.pem ]
4 then
5     date="$(date +%s)"
6     tempdir="/tmp/catemplate$date"
7     #echo "$tempdir"
8     mkdir -p "$tempdir"
9     echo -e "cn=COWCorp\nca\ncert_signing_key" > "$tempdir/ca.
        info"
10
11     mkdir -p /etc/pki/CA/private
12     certtool --generate-privkey > /etc/pki/CA/private/cakey.pem
13     certtool --generate-self-signed --load-privkey /etc/pki/CA/
        private/cakey.pem --template "$tempdir/ca.info" --outfile /
        etc/pki/CA/cacert.pem 2> /tmp/cacreate.log
14     rm -r "$tempdir"
15 fi
16 ssh "root@$host" "mkdir -p /etc/pki/CA/"
17 rsync "/etc/pki/CA/cacert.pem" "root@$host:/etc/pki/CA/cacert.pem
    "

```

Listing 6.3: cacert.sh

```

1 #!/bin/bash
2
3 host=$1
4 date="$(date +%s)"
5 tempdir="/tmp/catemplate$date"
6 mkdir -p "$tempdir"
7 echo -e "country=COW\nstate=COW\ntry\nlocality=COW\n\
    norganization=COWCorp\ncn=$host\ntls_www_client\
    nencryption_key\nsigning_key" > "$tempdir/client.info"
8
9 certtool --generate-privkey > "$tempdir/clientkey.pem"
10 certtool --generate-certificate --load-privkey "$tempdir/
    clientkey.pem" \
11     --load-ca-certificate /etc/pki/CA/cacert.pem --load-ca-privkey
        /etc/pki/CA/private/cakey.pem \
12     --template "$tempdir/client.info" --outfile "$tempdir/
        clientcert.pem"
13
14 ssh "root@$host" "mkdir -p /etc/pki/libvirt/private/"
15 rsync "$tempdir/clientkey.pem" "root@$host:/etc/pki/libvirt/
    private/clientkey.pem"
16 rsync "$tempdir/clientcert.pem" "root@$host:/etc/pki/libvirt/
    clientcert.pem"
17

```



```
18 rm -rf "$tempdir"
```

Listing 6.4: clientcert.sh

```
1 #!/bin/bash
2
3 host=$1
4 date="$(date +%s)"
5 tempdir="/tmp/catemplate$date"
6 mkdir -p "$tempdir"
7 echo -e "organization=COWCorp\ncn=$host\ntls_www_server\
nencryption_key\nsigning_key" > "$tempdir/server.info"
8
9 certtool --generate-privkey > "$tempdir/serverkey.pem"
10 certtool --generate-certificate --load-privkey "$tempdir/
serverkey.pem" \
11 --load-ca-certificate /etc/pki/CA/cacert.pem --load-ca-privkey
/etc/pki/CA/private/cakey.pem \
12 --template "$tempdir/server.info" --outfile "$tempdir/
servercert.pem" 2> "/var/log/cow.log"
13
14 ssh "root@$host" "mkdir -p /etc/pki/libvirt/private/"
15 rsync "$tempdir/serverkey.pem" "root@$host:/etc/pki/libvirt/
private/serverkey.pem"
16 rsync "$tempdir/servercert.pem" "root@$host:/etc/pki/libvirt/
servercert.pem"
17
18 rm -rf "$tempdir"
```

Listing 6.5: servercert.sh

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 import os
5 import re
6 import subprocess
7 import shutil
8 import sys
9 import libvirt
10 import socket
11 import string
12 import random
13 import pickle
14 import time
15 from xml.dom.minidom import parseString
```

```

16
17 class VHost:
18     a = "a"
19
20 def debugOut(output, debugLevel):
21     if debug >= debugLevel:
22         print 'Debug_[' + str(debugLevel) + ']:' + output
23
24 def randomName(vmName):
25     length = len(vmName) + 6
26     chars = string.letters+string.digits
27     name = vmName
28     while(len(name) < int(length)):
29         name += random.choice(chars)
30     #TODO? Sicherheitsabfrage
31     return name
32
33 def execute(command):
34     s = subprocess.Popen(command, stdout=subprocess.PIPE, stdin=
        subprocess.PIPE, stderr=subprocess.PIPE)
35     stdout, stderr = s.communicate()
36     debugOut(stderr, 1)
37     debugOut(stdout, 3)
38     return stdout
39
40 def prepareXml(xmlDescription):
41     hList = []
42     description = parseString(xmlDescription)
43     hardDisks = description.getElementsByTagName("disk")
44     for i in range(0, len(hardDisks)):
45         if hardDisks[i].getAttribute("device") == "disk":
46             hardDisk = hardDisks[i].getElementsByTagName("source")[0].
                getAttribute("file")
47             newHddPath = config.imageDir + "/" + newVmName + "-" + os.
                path.basename(hardDisk)
48             cloneHdd(hardDisk, newHddPath)
49             hardDisks[i].getElementsByTagName("source")[0].setAttribute
                ("file", newHddPath)
50     description.getElementsByTagName("name")[0].childNodes[0].data
        = newVmName
51
52     for removeTag in description.getElementsByTagName("mac"):
53         removeTag.parentNode.removeChild(removeTag)
54
55     for removeTag in description.getElementsByTagName("uuid"):

```

```

56     removeTag.parentNode.removeChild(removeTag)
57     return description
58
59 def cloneVm(vmName, vType):
60     if vType == "xen":
61         conn = libvirt.open('xen://')
62     else:
63         conn = libvirt.open('qemu:///system')
64     vm = conn.lookupByName(vmName)
65
66     #xmlFile = open(torrentDir + '/' + vmName + '.xml', 'w')
67     #xmlFile = open('/tmp/' + vmName + '.xml', 'w')
68     newVmXml = prepareXml(vm.XMLDesc(libvirt.VIR_DOMAIN_XML_SECURE)
69                             )
70     #save a temp copy, perhaps for debugging :)
71     #newVmXml.writexml(xmlFile)
72     #define the new VM in libvirt
73     vm = conn.defineXML(newVmXml.toxml())
74     if autostart:
75         vm.create()
76     #xmlFile.close()
77
78 def cloneHdd(hdd, newHddPath):
79     cloneMethod = {
80         'kvm' : cloneHddKvm,
81         'xen' : cloneHddXen,
82         'qemu' : cloneHddQemu}
83     cloneMethod[config.vType](hdd, newHddPath)
84
85 def cloneHddQemu(hdd, newHddPath):
86     command = ['qemu-img', 'info', hdd]
87     baseFormat = re.search('file_\u005Cformat:\u005C(?P<format>[\S]*)',
88                             execute(command)).groupdict()['format']
89     command = ['qemu-img', 'create', '-f', 'qcow2', '-b', hdd, '-o',
90                 'backing_fmt=' + baseFormat, newHddPath]
91     execute(command)
92
93 def cloneHddKvm(hdd, newHddPath):
94     command = ['kvm-img', 'info', hdd]
95     baseFormat = re.search('file_\u005Cformat:\u005C(?P<format>[\S]*)',
96                             execute(command)).groupdict()['format']
97     command = ['kvm-img', 'create', '-f', 'qcow2', '-b', hdd, '-o',
98                 'backing_fmt=' + baseFormat, newHddPath]
99     execute(command)

```

```

96 def cloneHddXen(hdd, newHddPath):
97     #TODO
98     command = ['vhd-util', 'snapshot', '-n', newHddPath, '-p', hdd]
99     execute(command)
100
101 if len(sys.argv) == 5:
102     startTimeSkript = time.time()
103     configPickle = open(os.path.expanduser('~/.config/whoami.pickle'), 'r')
104     config = pickle.load(configPickle)
105     configPickle.close()
106
107     vmName = sys.argv[1]
108     cloneCount = int(sys.argv[2])
109     if sys.argv[3] == 'y' or sys.argv[3] == 'Y' or sys.argv[3] == 'j' or sys.argv[3] == 'J':
110         autostart = 'y'
111     else:
112         autostart = None
113     debug = int(sys.argv[4])
114
115     for i in range(0, cloneCount):
116         startTimeClone = time.time()
117         newVmName = randomName(vmName)
118         cloneVm(vmName, config.vType)
119         print 'Klon_' + str(i) + ':' + newVmName
120         debugOut('Time_needed:' + str(time.time() - startTimeClone), 1)
121     debugOut('Overall_Time_needed:' + str(time.time() - startTimeSkript), 1)

```

Listing 6.6: client-scripts/clone.py

```

1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  import os
5  import re
6  import subprocess
7  import shutil
8  import sys
9  import libvirt
10 import socket
11 import pickle
12 from xml.dom.minidom import parseString
13

```

```

14 class VHost:
15     a = "a"
16
17 def execute(command):
18     s = subprocess.Popen(command, stdout=subprocess.PIPE, stdin=
19         subprocess.PIPE, stderr=subprocess.PIPE)
20     stdout, stderr = s.communicate()
21     if debug > 0:
22         print stdout
23         print stderr
24     return stdout
25
26 def hddList(xmlDescription):
27     hList = []
28     description = parseString(xmlDescription)
29     hardDisks = description.getElementsByTagName("disk")
30     for i in range(0, len(hardDisks)):
31         if hardDisks[i].getAttribute("device") == "disk":
32             hardDisk = hardDisks[i].getElementsByTagName("source")[0].
33                 getAttribute("file")
34             hList.append(hardDisk)
35             """hardDiskFile = hardDisks[i].getElementsByTagName("source
36                 ") [0].getAttribute("file")
37             hardDiskDev = hardDisks[i].getElementsByTagName("source")
38                 [0].getAttribute("dev")
39             if hardDiskFile:
40                 hList.append(hardDiskFile)
41             elif hardDiskDev:
42                 hList.append(hardDiskDev)"""
43     return hList
44
45 def prepareXml(xmlDescription, vmName):
46     hList = []
47     description = parseString(xmlDescription)
48     hardDisks = description.getElementsByTagName("disk")
49     for i in range(0, len(hardDisks)):
50         if hardDisks[i].getAttribute("device") == "disk":
51             hardDisk = hardDisks[i].getElementsByTagName("source")[0].
52                 getAttribute("file")
53             hardDisks[i].getElementsByTagName("source")[0].setAttribute
54                 ("file", imageDir + "/" + vmName + "/" + os.path.
55                     basename(hardDisk))
56     #remove uuid and mac address
57     removeTag = description.getElementsByTagName("mac")[0]
58     removeTag.parentNode.removeChild(removeTag)

```

```

52     removeTag = description.getElementsByTagName("uuid")[0]
53     removeTag.parentNode.removeChild(removeTag)
54     return description
55
56 def makeTorrent(vmName):
57     if vType == "xen":
58         conn = libvirt.open('xen://')
59     else:
60         conn = libvirt.open('qemu:///system')
61     vm = conn.lookupByName(vmName)
62
63     torrentDir = downloadDir + '/' + vm.name()
64     #make an empty torrent directory
65     if os.path.exists(torrentDir):
66         shutil.rmtree(torrentDir)
67     os.makedirs(torrentDir)
68
69     hdList = hddList(vm.XMLDesc(libvirt.VIR_DOMAIN_XML_SECURE))
70     for hd in hdList:
71         if not os.path.exists(hd):
72             print 'Festplatte' + hd + ' nicht vorhanden!\nVerteilen
              wird abgebrochen.'
73             sys.exit(1)
74     for hd in hdList:
75         os.symlink(hd, torrentDir + '/' + os.path.basename(hd))
76         print hd + "->" + torrentDir + '/' + os.path.basename(hd)
77     xmlFile = open(torrentDir + '/' + vmName + '.xml', 'w')
78     #xmlFile.write(modifiedHddList(vm.XMLDesc(libvirt.
              VIR_DOMAIN_XML_SECURE), vmName))
79     prepareXml(vm.XMLDesc(libvirt.VIR_DOMAIN_XML_SECURE), vmName).
              writexml(xmlFile)
80     xmlFile.close()
81     torrentFileName = config.downloadDir + "/" + vmName + '.torrent
              '
82     if os.path.exists(torrentFileName):
83         os.remove(torrentFileName)
84     command = ['mktorrent', '-a', config.ip, '-o' , torrentFileName
              , torrentDir]
85     execute(command)
86     #add the announce server as a dht node
87     torrentFile = open(torrentFileName, 'r')
88     announce = '8:announce' + str(len(config.ip)) + ':' + config.ip
89     addedDht = torrentFile.read().replace(announce , announce + '5:
              nodes11' + str(len(config.ip)) + ':' + config.ip + '
              i6881eee')

```

```

90     torrentFile.close()
91     torrentFile = open(torrentFileName, 'w')
92     torrentFile.write(addedDht)
93     torrentFile.close()
94
95     if not os.path.exists(os.path.expanduser('~/.config/')):
96         os.mkdir('~/.config/')
97     configPickle = open(os.path.expanduser('~/.config/whoami.pickle'),
98                        , 'r')
99     config = pickle.load(configPickle)
100    configPickle.close()
101    vmName = sys.argv[1]
102    downloadDir = config.downloadDir
103    imageDir = config.imageDir
104    vType = config.vType
105    debug = 1
106    makeTorrent(vmName)

```

Listing 6.7: client-scripts/maketorrent.py

```

1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  import re
5  import subprocess
6  import sys
7
8  def execute(command):
9      s = subprocess.Popen(command, stdout=subprocess.PIPE, stdin=
10                          subprocess.PIPE, stderr=subprocess.PIPE)
11      stdout, stderr = s.communicate()
12      print stdout
13      print stderr
14      return stdout
15
16  packages = re.split('[\s]+', sys.argv[1])
17
18  command = ['apt-get', 'install', '-y'] + packages
19  execute(command)

```

Listing 6.8: client-scripts/packageinstall.py

```

1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  import pickle

```

```

5 import os
6 import subprocess
7 import sys
8 import re
9
10 from socket import gethostname;
11
12 class VHost:
13     def __init__(self, ip, name, vType, imageDir, downloadDir):
14         self.name = name
15         self.ip = ip
16         self.vType = vType
17         self.imageDir = imageDir
18         self.downloadDir = downloadDir
19
20     def execute(command):
21         s = subprocess.Popen(command, stdout=subprocess.PIPE, stdin=
22             subprocess.PIPE)
23         #stdout, stderr = s.communicate()
24         #if debug > 0:
25         #print stdout
26         #print stderr
27         return s.communicate()
28
29 name = gethostname()
30 ip = sys.argv[1]
31 command = ['find', "/boot/", "-name", "xen*.gz"]
32 stdout = execute(command)
33
34 if stdout and re.match('[\S]*xen[\S]*', os.uname()[2]):
35     vType = "xen"
36 else:
37     vType = "kvm"
38
39 downloadDir = sys.argv[2]
40 imageDir = sys.argv[3]
41
42 obj = VHost(ip, name, vType, imageDir, downloadDir)
43 if not os.path.exists(os.path.expanduser('~/.config')):
44     os.mkdir(os.path.expanduser('~/.config'))
45 cowpickle = open(os.path.expanduser('~/.config/whoami.pickle'), '
46     w')
47 pickle.dump(obj, cowpickle)
48 cowpickle.close()

```


Listing 6.9: client-scripts/whoami.py

```

1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  import os
5  import re
6  import subprocess
7  import shutil
8  import sys
9  import libvirt
10 import socket
11 from xml.dom.minidom import parseString
12
13 configLines = open(os.path.expanduser('/etc/xen/xend-config.sxp')
14                   , 'r').readlines()
15
16 def httpNo():
17     for line in configLines:
18         if re.match('[\s]*\s(xend-http-server_\sno)\s[\s]*',line):
19             return
20
21     config = open('/etc/xen/xend-config.sxp','a')
22     config.write('\n(xend-http-server_\sno)\n')
23     config.close()
24
25 def unixYes():
26     for line in configLines:
27         if re.match('[\s]*\s(xend-unix-server_\syes)\s[\s]*',line):
28             return
29
30     config = open('/etc/xen/xend-config.sxp','a')
31     config.write('\n(xend-unix-server_\syes)\n')
32     config.close()
33
34 httpNo()
35 unixYes()

```

Listing 6.10: client-scripts/xenprep.py

Literaturverzeichnis

- [Bau] BAUN, Christian: *Vorlesung Systemsoftware*. http://jonathan.sv.hs-mannheim.de/~c.baun/SYS0708/Skript/folien_sys_vorlesung_13_WS0708.pdf, Abruf: 31.10.2010
- [Bro] BROŽ, Milan: *Device mapper*. <http://mbroz.fedorapeople.org/talks/DeviceMapperBasics/dm.pdf>, Abruf: 17.10.2010
- [CB05] CESATI, Marco ; BOVET, Daniel P.: *Understanding the Linux Kernel*. dritte Ausgabe. Linux-Server-Praxis, 2005
- [Coh08] COHEN, Bram: *The BitTorrent Protocol Specification*. http://www.bittorrent.org/beps/bep_0003.html. Version: 2008, Abruf: 01.01.2011
- [Cro] CROSBY, Simon: *We've Open Sourced Our Optimized VHD Support*. <http://community.citrix.com/x/0YKiAw>, Abruf: 11.11.2010
- [dmk] *Device-mapper snapshot support*. <http://www.kernel.org/doc/Documentation/device-mapper/snapshot.txt>, Abruf: 17.10.2010
- [EK] EGER, Kolja ; KILLAT, Ulrich: *Scalability of the BitTorrent P2P Application*. <http://www3.informatik.uni-wuerzburg.de/ITG/2005/presentations/kolja.eger.pdf>, Abruf: 01.01.2011

- [Ker00] KERR, Shane: *Use of NFS Considered Harmful*.
http://www.time-travellers.org/shane/papers/NFS_considered_harmful.html. Version: 2000, Abruf: 01.01.2011

- [Lei] LEITNER, Felix von: *Wir erfinden IP Multicasting*. <http://www.fefe.de/multicast/multicast.pdf>, Abruf: 01.01.2011

- [Loe08] LOEWENSTERN, Andrew: *DHT Protocol*. http://www.bittorrent.org/beps/bep_0005.html. Version: 2008, Abruf: 01.01.2011

- [lvma] *Linux LVM-HOWTO*. <http://www.selflinux.org/selflinux/html/lvm01.html>, Abruf: 18.10.2010

- [lvmb] *LVM2 Resource Page*. <http://sourceware.org/lvm2/>, Abruf: 18.10.2010

- [lvmc] *What is Logical Volume Management?* <http://tldp.org/HOWTO/LVM-HOWTO/whatisvolman.html>, Abruf: 18.10.2010

- [McL] MCLOUGHLIN, Mark: *The QCOW2 Image Format*. <http://people.gnome.org/~markmc/qcow-image-format.html>, Abruf: 18.10.2010

- [mso] *Microsoft Open Specification Promise*. <https://www.microsoft.com/interop/osp/default.aspx>, Abruf: 18.10.2010

- [mul] *Multicast FAQ File*. <http://www.multicasttech.com/faq/>, Abruf: 01.01.2011

- [nfs] *4. Setting up an NFS Client*. http://nfs.sourceforge.net/nfs-howto/ar01s04.html#mounting_remote_dirs, Abruf: 01.01.2011

- [nfs03] The Internet Society: *Network File System (NFS) version 4 Protocol*. <http://tools.ietf.org/html/rfc3530>. Version: 2003, Abruf: 01.01.2011
- [Prz] PRZYWARA, André: *Virtualization Primer*. <http://www.andrep.de/virtual/>, Abruf: 01.11.2010
- [qco] *Qcow2 Support*. <http://lists.xensource.com/archives/html/xen-devel/2010-11/msg00256.html>, Abruf: 14.11.2010
- [qem] *QEMU Emulator User Documentation*. http://wiki.qemu.org/download/qemu-doc.html#disk_005fimages, Abruf: 18.10.2010
- [rac] *Race condition in /etc/xen/scripts/block*. <http://lists.xensource.com/archives/html/xen-devel/2010-07/msg00827.html>, Abruf: 14.11.2010
- [spa] *Sparse files*. <http://www.lrdev.com/lr/unix/sparsefile.html>, Abruf: 18.10.2010
- [vhd] *Virtual Hard Disk Image Format Specification*. <http://technet.microsoft.com/en-us/virtualserver/bb676673.aspx>, Abruf: 18.10.2010
- [vmw] VMware: *VMware Benchmarking Approval Process*. http://www.vmware.com/pdf/benchmarking_approval_process.pdf, Abruf: 05.11.2010

Abbildungsverzeichnis

2.1	Copy-on-Write	9
2.2	Sparse-Datei	10
2.3	Iozone-kvm-8gb	15
2.4	bonnie-kvm-random-seek	15
2.5	Iozone-xen-8gb	16
2.6	bonnie-xen-random-seek	17
3.1	Multicast Beispiel	20
3.2	Bittorrent Beispiel	22
3.3	NFS Beispiel	23
3.4	Bittorrent Netzwerkausfall	25
3.5	Multicast Netzwerkausfall	26
3.6	NFS Netzwerkausfall	26
4.1	Kommunikation	32

Listings

4.1	libvirt-XML Beispiel	31
4.2	VHost-Auswahl	34
4.3	VM-Auswahl	35
4.4	modifizierte XML-Beschreibung	36
6.1	cow.py	40
6.2	hostname.sh	47
6.3	cacert.sh	47
6.4	clientcert.sh	48
6.5	servercert.sh	49
6.6	client-scripts/clone.py	49
6.7	client-scripts/maketorrent.py	52
6.8	client-scripts/packageinstall.py	55
6.9	client-scripts/whoami.py	55
6.10	client-scripts/xenprep.py	57