University of Applied Sciences
HOCHSCHULE EMDEN·LEER
Fachbereich Technik
Abteilung Elektrotechnik und Informatik

Bachelor-Thesis

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

Bastian de Groot

11. Januar 2011

Prüfer Prof. Dr. Jörg Thomaschewski

Zweitprüfer Dr. Arvid Requate

Inhaltsverzeichnis

1 Einleitung

Virtualisierung ist heute ein sehr wichtiger Begriff in der Informatik. Die Virtualisierung verteilt die vorhandenen Ressourcen effizient und hilft somit Prozesse zu optimieren und Kosten zu senken. Es gibt unterschiedliche Formen von Virtualisierung wie zum Beispiel die Anwendungsvirtualisierung und die Betriebsystemvirtualisierung. Diese Arbeit beschäftigt sich mit einem Aspekt der Betriebsystemvirtualisierung.

Von "Betriebsystemvirtualisierung" spricht man, wenn sich mehrere virtuelle Betriebsysteminstanzen Hardwareressourcen wie CPU, RAM oder Festplatten teilen. Der Virtualisierungskern (Hypervisor) stellt den virtuellen Betriebsysteminstanzen eine in Software und Hardware realisierte Umgebung zur Verfügung, die für die darin laufenden Instanzen kaum von einer echten Hardwareumgebung unterscheidbar sind [Prz] [Bau]. 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.

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 diesem 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 Kapitel 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 Lö-

sung ohne tiefgreifende Vorkenntnisse ermöglicht.

1.3 Anmerkung zur Verwendung von Open-Source

Für die Verwendung von Open-Source gibt es mehrere Gründe. Führende Hersteller von Virtualisierungssoftware wie zum Beispiel Citrix bieten in vielen Bereichen Lösungen an, die auf Open-Source-Technologien basieren. Auch zu beachten ist, dass der finanzielle Rahmen dieser Arbeit den Einsatz von proprietärer Software nicht ermöglicht. 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 Bewertbarkeit 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. Sie 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 ist, 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.

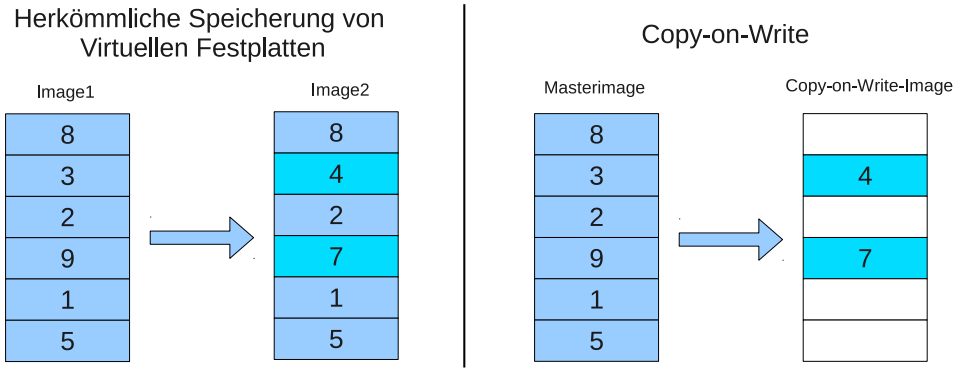

Abbildung 2.1: Copy-on-Write

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.

2.2 Imageformat qcow2

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

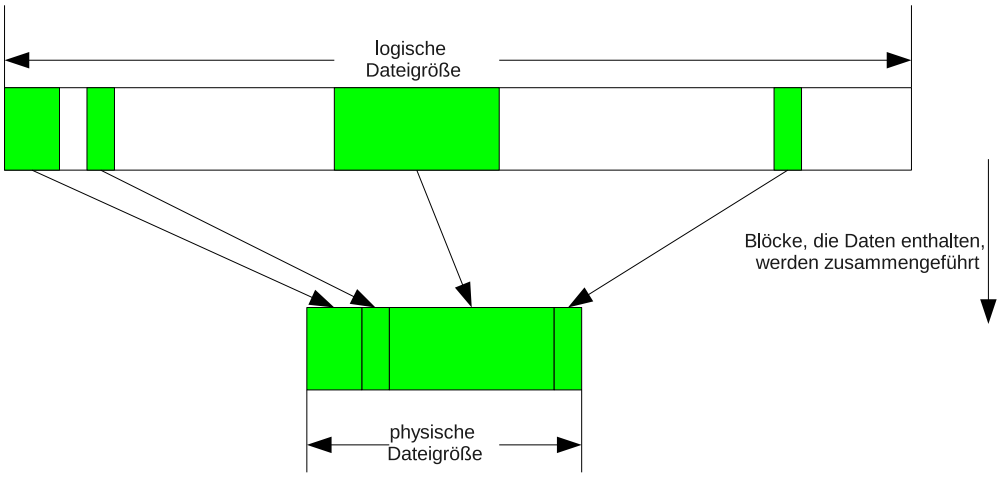

Abbildung 2.2: Sparse-Datei

stammenden Formats qcow [McL].

Vorteile

- Einfache Einrichtung
- Aktive Entwicklung im Rahmen der Projekte KVM und qemu

Nachteil

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

2.3 Imageformat 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 derzeit 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 zweiten 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 virtuelle 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.

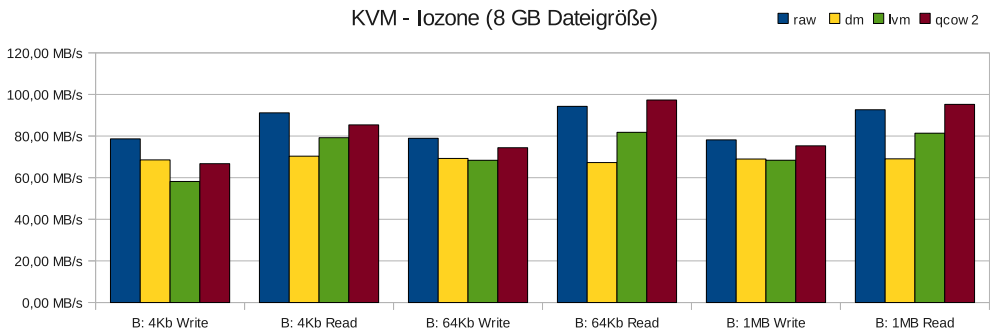

Abbildung 2.3: Performance-Testergebnisse von Iozone für KVM mit der Dateigröße 8gb

Die Abbildung 2.3 zeigt, dass mit KVM qcow2 gegenüber den anderen Copy-on-Write-Techniken einen Geschwindigkeitsvorteil beim sequentiellen Lesen und Schreiben hat. Insbesondere bei großen Blockgrößen zeigt

sich dieser Vorteil. LVM-Snapshots und dm-Snapshots liegen hingegen ungefähr gleich auf.

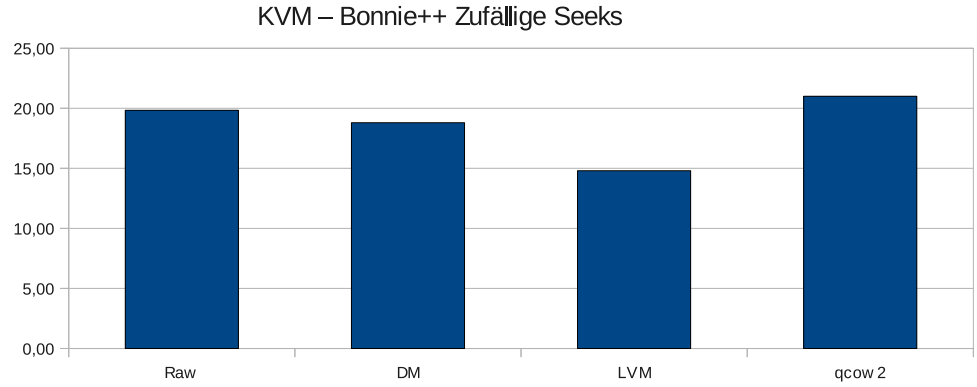

Abbildung 2.4: Performance-Testergebnisse von bonnnie++ für KVM

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 zufälligem 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.

Bei zufälligen Zugriff auf die Festplatte unter Xen ist VHD langsamer als LVM-Snapshots und dm-Snapshots. Die LVM-Snapshots und dm-Snapshots haben eine ähnliche Geschwindigkeit und keine signifikanten Nachteile gegenüber der Festplattenpartition (siehe Abbildung 2.6). Trotz der von Citrix für Xen eigens entwickelten VHD-Unterstützung, hat VHD nur ein mittelmäßiges Ergebnis erzielt [Cro].

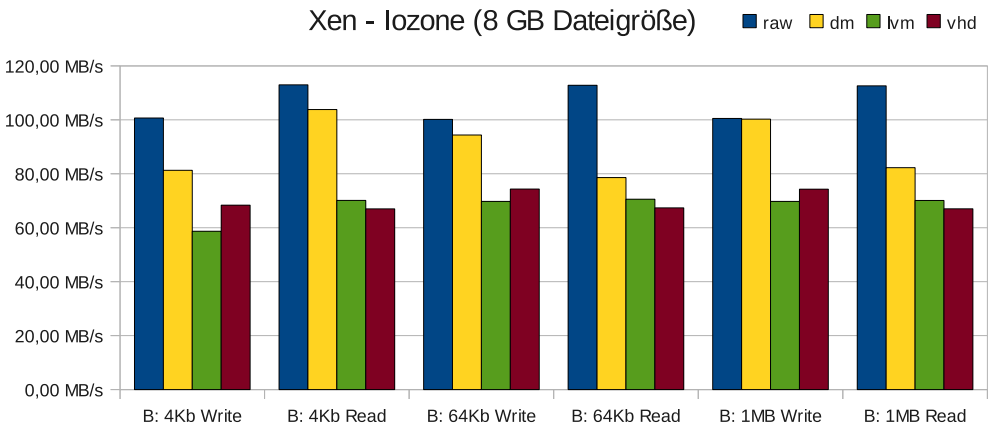

Abbildung 2.5: Performance-Testergebnisse von Iozone für Xen mit der Dateigröße 8gb

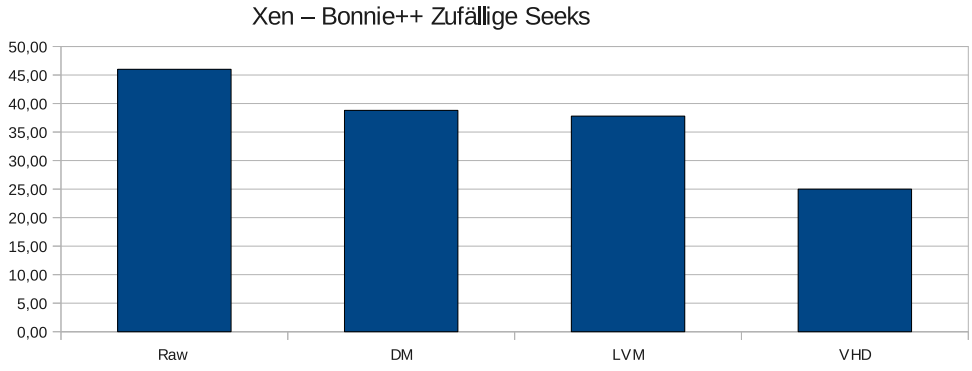

Abbildung 2.6: Performance-Testergebnisse von bonnnie++ Xen

Die Testergebnisse zeigen, dass es Geschwindigkeitsunterschiede zwischen den Copy-on-Write-Technniken gibt. Diese Unterschiede in der Geschwindigkeit sind aber nicht so groß, dass man einzelne Copy-on-Write-Lösungen aufgrund der Performance-Tests kategorisch ausschließen müsste. Dennoch sind besonders die Vorteile von qcow2 in Verbindung mit KVM zu erwähnen. Für Xen gibt es zur Zeit kein Image-Format (siehe Abbildung 2.5), dass ähnliche Testergebnisse wie qcow2 in Verbindung mit KVM (siehe Abbildung 2.4) vorweisen kann.

2.7 Fazit

Es gibt bei den Testergebnissen keine eindeutige Empfehlung für eine Copy-on-Write-Technik aufgrund der Geschwindigkeit. Im Großen und Ganzen fallen bei den Ergebnissen unter den einzelnen Copy-on-Write Verfahren Unterschiede auf, sie lassen jedoch keine eindeutige Entscheidung zu.

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, da KVM zwar das VHD-Format unterstützt, aber 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 4.0.1 enthalten [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, die 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 Netzwerkknoten [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].

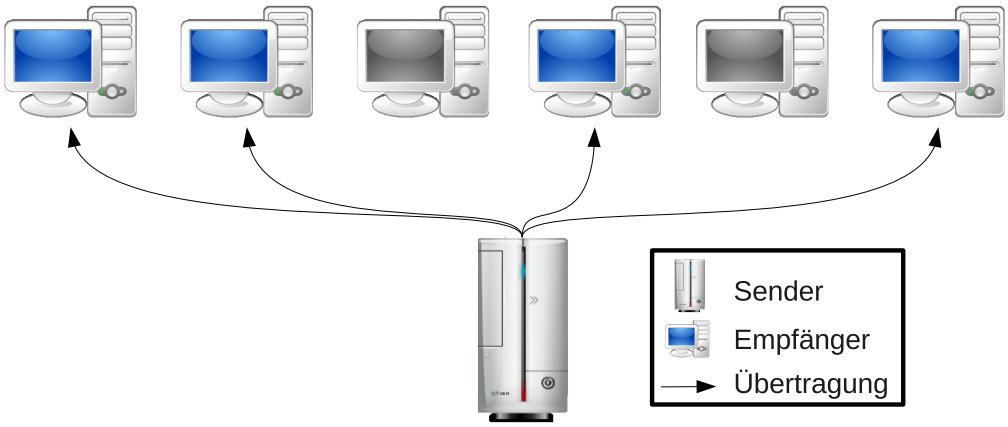

Abbildung 3.1: Multicast Beispiel

Vorteil

- Sehr hohe Geschwindigkeit durch Parallelität

Nachteile

- Hohe Netzwerklast ährend der Verteilung
- Geschwindigkeitseinbruch bei heterogener Umgebung oder schlechten Netzanbindungen

3.2 Netzwerkprotokoll BitTorrent

BitTorrent ist ein Netzwerkprotokoll zum effizienten Verteilen großer Dateien oder von Sammlungen großer 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 werden sogenannte Tracker eingesetzt. Es wird mindestens ein Tracker für die Kontaktaufnahme 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 Ausfallsichertheit, da die Liste der verfügbaren Peers dezentral gespeichert wird.

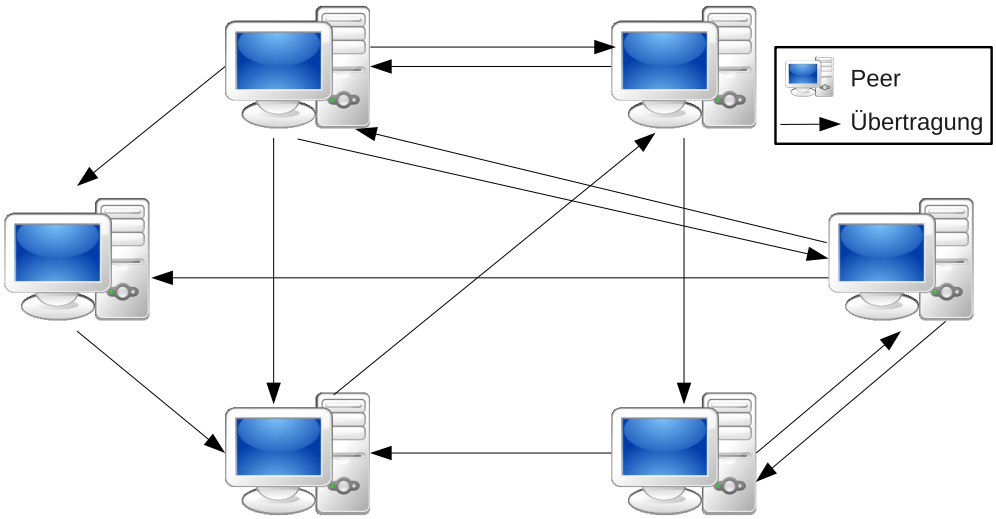

Abbildung 3.2: Bittorrent Beispiel

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

- verringerte Geschwindigkeit bei asymmetrischer Upload- und Downloadrate

3.3 Netzwerkprotokoll NFS

NFS (Network File System) ist ein Protokoll für das Bereitstellen von Daten über das Netzwerk. Im Gegensatz zu den beiden vorher genannten Technologien stellt dieses einen großen Unterschied dar. 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 den Verzeichnisbaum eingebunden und kann wie lokales Verzeichnis angesteuert werden.

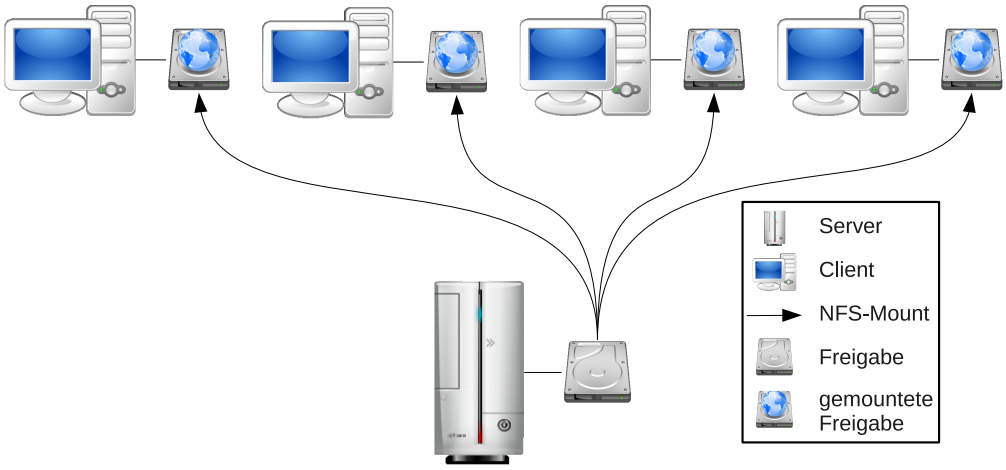

Abbildung 3.3: NFS Beispiel

Vorteile

- Geringer Administrationsaufwand

Nachteile

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

3.4 Vergleich

Im Folgenden werden 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 (N+1) verteilt wird. NFS zeigt dabei eine Schwäche, die Last steigt des NFS-Servers stetig mit jedem neuen NFS-Client linear 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 die Upload- und die Downloadrate gleich hoch sind, wird das Netz durch einen zusätzlichen Peer 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 gegen 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 großen Nachteil, da die Verteilung über einen einzigen Knoten stattfindet. Der Ausfall eines bestimmten Knotens führt also zum kompletten Abbruch

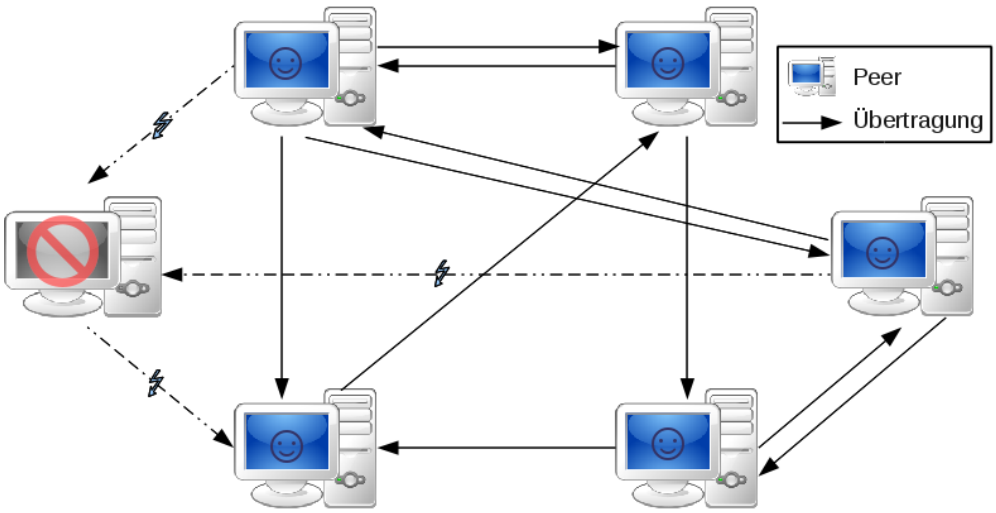

Abbildung 3.4: Bittorrent Netzwerkausfall

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 beteiligten Netzwerkknoten abhängig. Ideal ist es, wenn alle Empfänger und

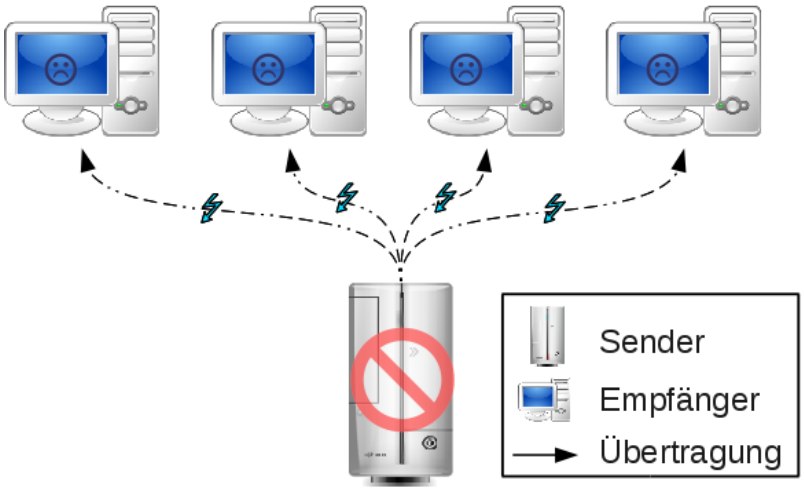

Abbildung 3.5: Multicast Netzwerkausfall

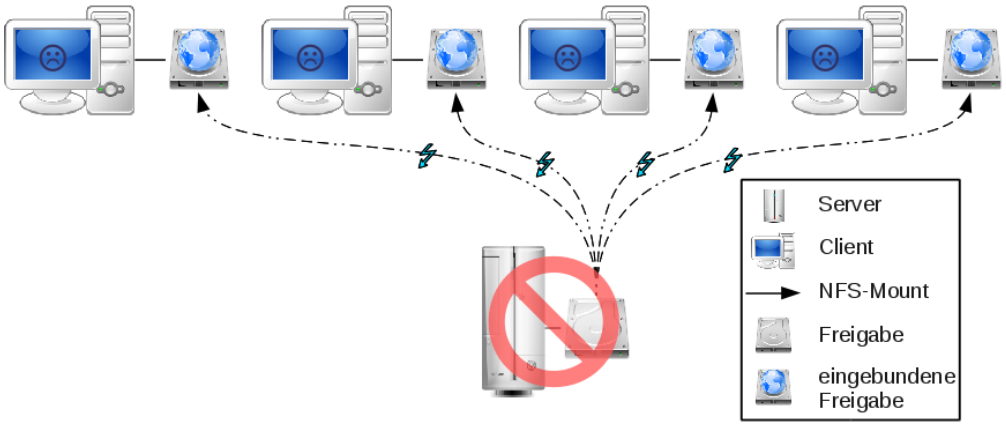

Abbildung 3.6: NFS Netzwerkausfall

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 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 Konzeptentwicklung und Realisierung

Die Erkenntnisse des vorhergehenden Kapitels werden in diesem Kapitel aufgegriffen und zu einem Konzept für die Implementierung zusammengefügt. Dieses Konzept wird im Anschluss auf geeignetem Wege umgesetzt.

4.1 Konzept

Die zu entwickelnde Verwaltungslösung dient dem Zweck virtuelle Maschinen sehr schnell zu replizieren und die dafür benötigten Vorlagen schnell auf die Virtualisierungsserver zu verteilen. Die nötigen Voraussetzungen dafür sind:

- Steuerung und Kommunikation
- Verteilung
- Klonen.

Im Folgenden werden diese Voraussetzungen im Einzelnen erläutert.

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 Implementierung

In diesem Unterkapitel werden die oben genannten Punkte aufgegriffen und deren Umsetzung beschrieben. Im einzelnen wird hier auf die Punkte wie die Verwaltung der virtuellen Maschinen, das Klonen und die Verteilung eingegangen. Eine Auflistung aller entwickelten Skripte findet sich im Anhang (siehe Kapitel 6.1).

4.2.1 Rahmenbedingungen

Die Komplettlösung wird auf einem Debian squeeze System implementiert. In der Implementierung 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 Listing 4.1 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 Softwarelösung 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.

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 IP-Adresse des Virtualisierungshosts an. Ebenfalls wird die Virtualisierungstechnik des neuen Hosts abgefragt. Nach der Eingabe wird über die IP-Adresse der Rechnername erfragt. Für die einfache Kommunikation wird der ssh-key des Verwaltungsservers auf dem zu verwaltenden Virtualisierungshost hinzugefügt (siehe Listing 4.2).

```
1 #!/bin/bash
2 ip="$1"
3 host=$(ssh root@$ip 'echo "$HOSTNAME"')
4 ssh-copy-id root@$host > /dev/null
```

Listing 4.2: Abruf des Rechnernamens und Kopieren des ssh-keys (hostname.sh)

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 (siehe Listing 4.3).

```
1 command = ['rsync', '-r', 'client-scripts/' ,'root@' + hostName +
      ':' + binDir]
2 execute(command)
```

Listing 4.3: Übertragung der Client-Skripte

Im Anschluss folgt die Installation der benötigten Software-Pakete. Es werden die Pakete für libvirt, deluge, sowie für administrative Tools installiert, wie das Listing 4.3 zeigt.

```
1 command = ['ssh', 'root@' + hostName, '/opt/cow/packageinstall.py
    "deluged deluge-console mktorrent libvirt-bin python-libvirt
    "' ]
2 execute(command)
```

Listing 4.4: Paketinstallation auf dem Virtualisierungsserver

Für die Abfrage über das Netzwerk verwendet libvirt Zertifikate. Es gibt drei unterschiedliche Zertifikate. Das Server-Zertifikat dient dazu die Echtheit des Virtualisierungsservers 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 anderen beiden Zertifikate zu erstellen und zu zertifizieren. Das Server-Zertifikat wird von dem Verwaltungsserver erstellt. Nach der Erstellung wird es, wie in Listing 4.5 zu sehen, auf dem Virtualisierungsserver abgelegt. (**Hinweis:** Dies ist nur eine vereinfachte Darstellung des Zertifikate-Systems. Eine ausführliche Beschreibung ist unter [lib] zu finden.)

```
1 certtool --generate-privkey > "$tempdir/serverkey.pem"
2 certtool --generate-certificate --load-privkey "$tempdir/
    serverkey.pem" \
3   --load-ca-certificate /etc/pki/CA/cacert.pem --load-ca-privkey
      /etc/pki/CA/private/cakey.pem \
4   --template "$tempdir/server.info" --outfile "$tempdir/
      servercert.pem" 2> "/var/log/cow.log"
5
6 ssh "root@$host" "mkdir -p /etc/pki/libvirt/private/"
7 rsync "$tempdir/serverkey.pem" "root@$host:/etc/pki/libvirt/
    private/serverkey.pem"
8 rsync "$tempdir/servercert.pem" "root@$host:/etc/pki/libvirt/
    servercert.pem"
```

Listing 4.5: Erstellung des Serverzertifikats für den Virtualisierungsserver

Die verwalteten Virtualisierungsserver werden in als Liste in einer Klartextdatei 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 Virtualisierungshost aus, der die zu verteilende virtuelle Maschine beherbergt. Die Virtualisierungshosts werden aus der zuvor abgelegten Klartextdatei ausgelesen (siehe Listing 4.6).

```
1 def hostList():
2   hostList = []
3   hosts = open(os.path.expanduser('~/.cow/vhosts'), 'r').
        readlines()
4   for i in range(0,len(hosts)):
5     host = hosts[i].split('\t')
6     if len(host) == 2: #ignore malformed rows
7       hostList.append([len(hostList), host[0], host[1].replace('\
            n','')])
8   return hostList
```

Listing 4.6: Auslesen der registrierten Virtualisierungshosts

Die Methode für die Auswahl lässt den Benutzer zwischen allen ausgeschalteten virtuellen Maschinen auswählen. Diese werden wie im folgenden Listing 4.7 über die Virtualisierungs-API abgerufen.

```
1 def vmOffList(hostName, vType):
2   vOffList = []
3   if vType == 'xen':
4     conn = libvirt.open('xen://' + hostName + '/')
5   else:
6     conn = libvirt.open('qemu://' + hostName + '/system')
7
8   for name in conn.listDefinedDomains():
9     vOffList.append(conn.lookupByName(name))
```

```
10
11    return vOffList
```

Listing 4.7: Abruf der ausgeschalteten virtuellen mit libvirt

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 Kapitel 4.2.3) eine Torrent-Datei aus der XML-Beschreibung von libvirt und den virtuellen Festplatten. Sie wird an alle ausgewählten Virtualisierungsserver mit rsync weitergegeben.

Zuletzt werden alle BitTorrent-Clients gestartet und die erstellte torrent-Datei hinzugefügt. Durch das Hinzufügen wird automatisch der Download bzw. die Verteilung gestartet.

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 Listing **??**). Zusätzlich dazu wird die Anzahl der Klone und die Option alle Klone sofort zu starten abgefragt. Nach den erfolgten Be-

nutzereingaben ruft das Verwaltungsprogramm das auf dem Virtualisierungsserver befindliche Skript `clone.py` zum Klonen auf.

```
command = ['ssh', 'root@' + hostName, binDir + '/clone.py ' + vm.
    name() + ' ' + cloneCount + ' ' + autostart + ' ' + str(debug
    )]
stdout, stderr = execute(command)
```

Listing 4.8: Starten des Klonvorgangs

Im ersten Schritt des Klonvorgangs generiert das Skript einen neuen Namen (siehe Listing 4.9). Der Name setzt sich aus dem alten Namen und 6 zufälligen und Buchstaben zusammen.

```
def randomName(vmName):
  length = len(vmName) + 6
  chars = string.letters+string.digits
  name = vmName
  while(len(name) < int(length)):
    name += random.choice(chars)
  return name
```

Listing 4.9: Erstellen des Namens der VM

Der nächste Schritt ist es die Beschreibung der Vorlage aus libvirt zu laden. Aus ihr werden die Festplatten der Vorlage ausgelesen und geklont. Wie in dem folgenden Listing 4.10 gezeigt wird, gibt es unterschiedliche Klon-Methoden für Xen und KVM. Sie rufen die Klonwerzeuge der jeweiligen Virtualisierungstechnik auf.

```
 def cloneHddKvm(hdd, newHddPath):
  command = ['kvm-img', 'info' , hdd]
  baseFormat = re.search('file format: (?P<format>[\S]*)',
      execute(command)).groupdict()['format']
  command = ['kvm-img', 'create', '-f', 'qcow2', '-b', hdd, '-o',
      'backing_fmt=' + baseFormat, newHddPath]
  execute(command)

```

```
7 def cloneHddXen(hdd, newHddPath):
8   command = ['vhd-util', 'snapshot', '-n', newHddPath, '-p', hdd]
9   execute(command)
```

Listing 4.10: VM-Auswahl

Die Identifikationsnummer und die MAC-Adresse aus der XML-Beschreibung werden gelöscht und der neue Name eingetragen. Die MAC-Adresse und die Identifikationsnummer generiert libvirt neu beim Anlegen der geklonten virtuellen Maschine. Die Änderungen an der XML-Beschreibung sind in dem Listing 4.11 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    <devices>
11      <emulator>/usr/bin/kvm</emulator>
12      <disk type='file' device='disk'>
13       <driver name='qemu' type='qcow2'/>
14 -      <source file='/var/lib/libvirt/images/debian.qcow2'/>
15 +      <source file='/var/lib/libvirt/images/debianVxyIZ5-debian.
       qcow2'/>
16        <target dev='hda'/>
17      </disk>
18      <interface type='network'>
19 -      <mac address='52:54:00:3d:eb:4b'/>
20 +
21        <source network='default'/>
22      </interface>
23      <input type='mouse' bus='ps2'/>
24      <graphics type='vnc' port='-1' listen='0.0.0.0'/>
25    </devices>
```

Listing 4.11: modifizierte XML-Beschreibung

4.2.6 Gesamtübersicht der Implementierung

Die Softwarelösung unterscheidet grundsätzlich zwischen zwei unterschiedlichen Knotenpunkten. Zum einen gibt es den Virtualisierungsserver oder auch Virtualisierungshost. Er beherbergt die virtuellen Maschinen. Zum anderen gibt es den Verwaltungsserver. Auf ihm findet die Verwaltung statt und es wird von dem Verwaltungsserver ausgehend das Klonen und das Verteilen der virtuellen auf den Virtualisierungshosts initiiert (siehe Abbildung 4.1).

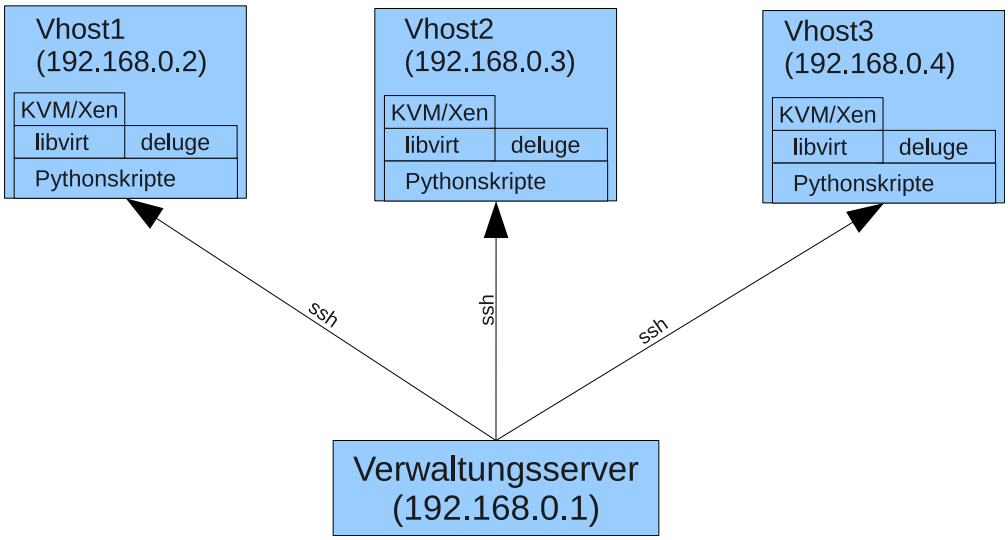

Abbildung 4.1: Kommunikation

Die aufwendigeren Aktionen wie das Klonen und erstellen der Torrent-Datei werden nicht über das Netzwerk vom Virtualisierungsserver erledigt. Statt dessen sind sie in Skripte auf dem Virtualisierungsservern ausgelagert. Der Virtualisierungsserver übernimmt das sammeln der Parameter und übergibt sie den lokalen Skripten beim Starten per ssh.

5 Zusammenfassung und Ausblick

In dem folgenden Kapitel wird ein Schlussfolgerungen über die erzielten Ergebnisse der Analyse gezogen und ein Fazit der Implementierung dargelegt. Abschließend werden in einem Ausblick mögliche Ansätze für die weitere Entwicklung der Implementierung gegeben.

5.1 Zusammenfassung

Es konnte mit objektiven Analysemethoden gezeigt werden, dass qcow2 und BitTorrent die geeignetsten Technologien für das schnelle Replizieren und Verteilen von virtuellen Maschinen sind. Mit der an die Analyse anschließenden Implementierung wurden diese so zur Verfügung gestellt, dass sie auch ohne Vorkenntnisse bedienbar sind. Außerdem wurde aufgezeigt, dass die analysierten Technologien praxistauglich sind.

Das Imageformat wird VHD im Gegensatz zu den beiden oben genannten Technologien nur als zum Einsatz kommen. Es ist zwar praxistauglich, jedoch hat es Performancenachteile im Vergleich zu qcow2. Die Schritt wurde notwendig da es während der Xen-Entwicklung zu einer fragwürdigen Umstrukturierung kam. Die Umstrukturierung führt dazu, dass Xen bestimmte Imageformate wie qcow2 nicht mehr unterstützt. Dieser Punkt

und die Tatsache das Xen nicht ohne selbst eingespielte Patches fehlerfrei einsetzbar ist, führen zu dem Urteil, dass der Einsatz von Xen nicht zu empfehlen ist.

Die entwickelte Softwarelösung ermöglicht das Replizieren und Verteilen. Somit hat die Softwarelösung die gestellten Erwartungen, virtuelle Maschinen schnell zu Klonen und über das Netzwerk zu verteilen, erfüllt. Hierbei war der Einsatz von bereits vorhandener Open Source Software sehr hilfreich. Ohne die eingebundene freie Software wäre dieses Projekt zeitlich nicht in dem Rahmen dieser Arbeit realisierbar gewesen.

In dieser Arbeit wird der wichtige Punkt deutlich, dass bei einer Softwareentwicklung nicht immer das Rad neu erfunden werden muss. Dieses gilt vor allem im Bereich von Open Source. Hier gibt es bereits sehr viele freie Bibliotheken und Programme für alle Themengebiete. Sie können problemlos in einer Eigenentwicklung verwendet und eingebunden werden. Trotz der nicht vorhandenen Lizenzkosten, ist Open Source Software in der Regel nicht funktionell eingeschränkter oder langsamer als proprietärer Software.

5.2 Ausblick

Für die weitere Entwicklung wäre es möglich die Bedienung für den Benutzer durch eine grafische Oberfläche zu vereinfachen. Hierfür könnte das grafische QT eingesetzt werden, dass auch Bibliotheken für die in der Implementierung verwendete Sprache Python bereitstellt.

Eine andere denkbare Weiterentwicklung wäre die Integration in libvirt. Hierbei könnten dann direkt Klone mit der Virtualisierungs-API erstellt werden.

6 Anhang

6.1 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.

6.1.1 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

6.1.2 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 klont 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

test

6.2 Code-Listings

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import os
import re
import subprocess
import shutil
import sys
import libvirt
import socket
import time
from xml.dom.minidom import parseString

vHostType = {1: 'kvm', 2: 'xen'}
downloadDir = '/var/lib/download'
imageDir = '/var/lib/libvirt/images'
binDir = '/opt/cow'

def debugOut(output,debugLevel):
  if debug >= debugLevel:
    print 'Debug [' + str(debugLevel) + ']:' + output

def hostList():
  hostList = []
  hosts = open(os.path.expanduser('~/.cow/vhosts'), 'r').
      readlines()
  for i in range(0,len(hosts)):
    host = hosts[i].split('\t')
    if len(host) == 2: #ignore malformed rows
      hostList.append([len(hostList), host[0], host[1].replace('\
          n','')])
  return hostList

def vmList(hostName, vType):
  try:
    vmList = vmOffList(hostName, vType) + vmOnList(hostName,
        vType)
    return vmList
  except libvirt.libvirtError:
    print 'Host ' + hostName + '(' + vType + ') nicht erreichbar'


def vmOnList(hostName, vType):
```

```
41    vOnList = []
42    if vType == 'xen':
43      conn = libvirt.open('xen://' + hostName + '/')
44    else:
45      conn = libvirt.open('qemu://' + hostName + '/system')
46
47    for id in conn.listDomainsID():
48      vOnList.append(conn.lookupByID(id))
49
50    return vOnList
51
52
53  def vHostExists(hostName):
54    hList = hostList()
55    for host in hList:
56      if host[1] == hostName:
57        return True
58    return False
59
60  def newVServer():
61    print 'Geben Sie die IP des Hosts ein'
62    ip = raw_input('HostIP: ')
63
64    print 'Virtualisierungstechnik'
65    print ' 1: KVM'
66    print ' 2: XEN'
67    choice = intInput('')
68
69    command = ['./hostname.sh', ip]
70    hostName, stderr = execute(command)
71    hostName = hostName.replace('\n', '')
72
73    if vHostExists(hostName):
74      print 'der Host ' + hostName + ' ist schon registriert'
75    else :
76      #append the new host
77      vhosts = open(os.path.expanduser('~/.cow/vhosts'), 'a')
78      vhosts.write('\n' + hostName + '\t' + vHostType[choice])
79      vhosts.close()
80
81      command = ['rsync', '-r', 'client-scripts/' ,'root@' +
            hostName + ':' + binDir]
82      execute(command)
83
```

```
84      command = ['ssh', 'root@' + hostName, '/opt/cow/
            packageinstall.py "deluged deluge-console mktorrent
            libvirt-bin python-libvirt"' ]
85      execute(command)
86
87      command = ['./cacert.sh', hostName]
88      execute(command)
89
90      command = ['./servercert.sh', hostName]
91      execute(command)
92
93      #command = ['./clientcert.sh', hostName]
94      #execute(command)
95
96      command = ['ssh', 'root@' + hostName, 'mkdir -p ' + binDir ]
97      execute(command)
98
99      command = ['ssh', 'root@' + hostName, binDir + '/whoami.py '
            + ip + ' ' + downloadDir + ' ' + imageDir ]
100     execute(command)
101
102 def clone():
103   hostName, vType = chooseVHost()
104   if hostName:
105     try:
106       vm = chooseVm(hostName,vType)
107       if vm:
108         cloneCount = raw_input('Anzahl der Klone: ')
109         autostart = raw_input('Nach Erstellung starten? (j/n): ')
110         command = ['ssh', 'root@' + hostName, binDir + '/clone.py
                ' + vm.name() + ' ' + cloneCount + ' ' + autostart +
                 ' ' + str(debug)]
111         stdout, stderr = execute(command)
112         print stdout
113       else:
114         print 'keine passende VM vorhanden'
115     except libvirt.libvirtError:
116       print 'Host ' + hostName + '(' + vType + ') nicht
              erreichbar'
117
118 def overview():
119   hList = hostList()
120   if hList:
121     for host in hList:
122       vList = vmList(host[1], host[2])
```

```
123       if vList:
124         print '\nHost: ' + host[1] + ' (' + host[2] + ')\n'
125         print 'ID\tName'
126         for vm in vList:
127           print vm.name() + '\t' + str(vm.info()[0])
128
129 def initialize():
130   if not os.path.exists(os.path.expanduser('~/.cow/')):
131     os.mkdir(os.path.expanduser('~/.cow/'))
132
133   if not os.path.exists(os.path.expanduser('~/.cow/vhosts')):
134     open(os.path.expanduser('~/.cow/vhosts'),'w').close()
135
136   if not os.path.exists('/var/log/cow.log'):
137     open('/var/log/cow.log', 'w').close()
138
139 def execute(command):
140   s = subprocess.Popen(command, stdout=subprocess.PIPE, stdin=
        subprocess.PIPE, stderr=subprocess.PIPE)
141   stdout, stderr = s.communicate()
142   debugOut(stdout, 3)
143   debugOut(stderr, 1)
144   return stdout, stderr
145
146 def hddList(xmlDescription):
147   hList = []
148   description = parseString(xmlDescription)
149   hardDisks = description.getElementsByTagName('disk')
150   for i in range(0,len(hardDisks)):
151     if hardDisks[i].getAttribute('device') == 'disk':
152       hardDisk = hardDisks[i].getElementsByTagName('source')[0].
            getAttribute('file')
153     hList.append(hardDisk)
154   return hList
155
156 def chooseVHost():
157   hList = hostList()
158   if hList:
159     print 'Wählen Sie den Virtualisierungshost aus:'
160     print 'ID\tHost\tTyp'
161     for host in hList:
162       print str(host[0]) + '\t' + host[1] + '\t' + host[2]
163     hostId = intInput('ID: ')
164     return [hList[hostId][1], hList[hostId][2]]
165   else:
```

```
166     print 'kein␣Virtualisierungshost␣vorhanden'
167     return [None,None]
168 
169 def chooseVHosts(filterName,filterVType):
170   print 'Verteilen␣an␣Hosts:'
171   hList = hostList()
172   for host in hList:
173     if host[2] == filterVType:
174       print str(host[0]) + '\t' + host[1] + '\t' + host[2]
175   targetHostIds = raw_input('IDs␣(z.B␣0,2,3):␣')
176   #remove whitespaces and split the comma seperated value
177   targetHostIds = targetHostIds.replace('\t', '').replace('␣', ''
        ).split(',')
178   #remove duplicates
179   targetHostIds = list(set(targetHostIds))
180   targetHosts = []
181   for i in targetHostIds:
182     if hList[int(i)][1] != filterName:
183       targetHosts.append(hList[int(i)])
184   return targetHosts
185 
186 def chooseVm(hostName,vType):
187   vOffList = vmOffList(hostName,vType)
188   if vOffList:
189     print 'Wählen␣Sie␣eine␣VM␣aus:'
190     print 'ID\tName\tState'
191     for i in range(0,len(vOffList)):
192       print str(i) + '\t' + vOffList[i].name() + '\t' + str(
            vOffList[i].info()[0])
193     vmId = intInput('ID:␣')
194     return vOffList[vmId]
195 
196 def startDownload(vHostList, vmName):
197   debugOut('start␣the␣deluge␣daemons␣on␣target␣hosts', 1)
198   #start the deluge daemon
199   for targetHost in vHostList:
200     debugOut('starting␣deluge␣daemon␣on' + targetHost[1], 1)
201     command = ['ssh', 'root@' + targetHost[1], 'deluged']
202     s = subprocess.Popen(command)
203   #waiting for the deluge daemon
204   time.sleep(1)
205   for targetHost in vHostList:
206     debugOut('rsync␣to␣' + targetHost[1], 2)
207     command = ('rsync', '/tmp/' + vmName + '.torrent', 'root@' +
          targetHost[1] + ':' + downloadDir + '/' + vmName + '.
```

```
        torrent')
208     debugOut(repr(command),3)
209     execute(command)
210     command = ['rsync', '/tmp/' + vmName + '.xml', 'root@' +
            targetHost[1] + ':/tmp/']
211     execute(command)
212     command = ['ssh', 'root@' + targetHost[1],  'virsh␣define␣/
            tmp/' + vmName + '.xml']
213     execute(command)
214
215     debugOut('del␣torrent␣on␣' + targetHost[1], 3)
216     command = ['ssh', 'root@' + targetHost[1], 'deluge-console␣"
            del␣' + vmName + '"']
217     s = subprocess.Popen(command)
218   #waiting for deletion of the old torrent
219   time.sleep(0.5)
220   #add the torrent and start downloading
221   for targetHost in vHostList:
222     debugOut('start␣torrent␣on' + targetHost[1],3)
223     command = ['ssh', 'root@' + targetHost[1], 'deluge-console␣"
            add␣' + downloadDir + '/' + vmName + '.torrent"']
224     s = subprocess.Popen(command)
225
226
227 def shareImage():
228   hostName, vType = chooseVHost()
229   if hostName:
230     try:
231       vm = chooseVm(hostName,vType)
232       if vm:
233         targetHosts = chooseVHosts(hostName,vType)
234         targetHosts.append([-1, hostName, vType])
235         command = ['ssh', 'root@' + hostName, '/opt/cow/
              maketorrent.py␣' + vm.name()]
236         stdout, stderr = execute(command)
237         if stderr:
238           print "Fehler:␣" + stderr
239           return
240         command = ['rsync', 'root@' + hostName + ':' +
              downloadDir + '/' + vm.name() + '.torrent', '/tmp/' ]
241         execute(command)
242         command = ['rsync', 'root@' + hostName + ':' +
              downloadDir + '/' + vm.name() + '/' + vm.name() + '.
              xml', '/tmp/' ]
243         execute(command)
```

```
244
245             startDownload(targetHosts, vm.name())
246           else:
247             print 'keine VM vorhanden'
248         except libvirt.libvirtError:
249           print 'Host ' + hostName + '(' + vType + ') nicht
                erreichbar'
250
251 def test():
252   print 'test'
253
254 def intInput(output):
255   tmpString = raw_input(output)
256   try:
257     intNum = int(tmpString)
258   except ValueError:
259     print 'Keine gültige Zahl'
260     intNum = intInput(output)
261   return intNum
262
263 options = {
264  1 : newVServer,
265  2 : shareImage,
266  3 : clone,
267  4 : overview,
268  0 : test}
269
270 initialize()
271
272 #set debugLevel
273 if len(sys.argv) > 1:
274   try:
275     debug = sys.argv[1]
276   except ValueError:
277     debug = 0
278 else:
279   debug = 0
280
281 while (True):
282   print 'Menü'
283   print ' 1: neuer Virtualisierungsserver'
284   print ' 2: Image verteilen'
285   print ' 3: Virtuelle Maschine klonen'
286   print ' 4: Übersicht aller virtuellen Maschinen'
287   print ' 5: Beenden'
```

```
288
289   choice = intInput('Auswahl: ')
290
291   if choice <= 4 and choice >= 0:
292     options[choice]()
293   else:
294     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 "$tempdir"
9    echo -e "cn = COW Corp\nca\ncert_signing_key" > "$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

```
#!/bin/bash

host=$1
date="$(date␣+%s)"
tempdir="/tmp/catemplate$date"
mkdir -p "$tempdir"
echo -e "country␣=␣COW\nstate␣=␣COWntry\nlocality␣=␣COWn\
    norganization␣=␣COW␣Corp\ncn␣=␣$host\ntls_www_client\
    nencryption_key\nsigning_key" > "$tempdir/client.info"

certtool --generate-privkey > "$tempdir/clientkey.pem"
certtool --generate-certificate --load-privkey "$tempdir/
    clientkey.pem" \
  --load-ca-certificate /etc/pki/CA/cacert.pem --load-ca-privkey
     /etc/pki/CA/private/cakey.pem \
  --template "$tempdir/client.info" --outfile "$tempdir/
     clientcert.pem"

ssh "root@$host" "mkdir␣-p␣/etc/pki/libvirt/private/"
rsync "$tempdir/clientkey.pem" "root@$host:/etc/pki/libvirt/
    private/clientkey.pem"
rsync "$tempdir/clientcert.pem" "root@$host:/etc/pki/libvirt/
    clientcert.pem"

rm -rf "$tempdir"
```

Listing 6.4: clientcert.sh

```
#!/bin/bash

host=$1
date="$(date␣+%s)"
tempdir="/tmp/catemplate$date"
mkdir -p "$tempdir"
echo -e "organization␣=␣COW␣Corp\ncn␣=␣$host\ntls_www_server\
    nencryption_key\nsigning_key" > "$tempdir/server.info"

certtool --generate-privkey > "$tempdir/serverkey.pem"
certtool --generate-certificate --load-privkey "$tempdir/
    serverkey.pem" \
  --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   return name
31
32 def execute(command):
33   s = subprocess.Popen(command, stdout=subprocess.PIPE, stdin=
```

```
      subprocess.PIPE, stderr=subprocess.PIPE)
  stdout, stderr = s.communicate()
  debugOut(stderr, 1)
  debugOut(stdout, 3)
  return stdout

def prepareXml(xmlDescription):
  hList = []
  description = parseString(xmlDescription)
  hardDisks = description.getElementsByTagName("disk")
  for i in range(0,len(hardDisks)):
    if hardDisks[i].getAttribute("device") == "disk":
      hardDisk = hardDisks[i].getElementsByTagName("source")[0].
          getAttribute("file")
      newHddPath = config.imageDir + "/" + newVmName + "-" + os.
          path.basename(hardDisk)
      cloneHdd(hardDisk, newHddPath)
      hardDisks[i].getElementsByTagName("source")[0].setAttribute
          ("file", newHddPath)
  description.getElementsByTagName("name")[0].childNodes[0].data
      = newVmName

  for removeTag in description.getElementsByTagName("mac"):
    removeTag.parentNode.removeChild(removeTag)

  for removeTag in description.getElementsByTagName("uuid"):
    removeTag.parentNode.removeChild(removeTag)
  return description

def cloneVm(vmName, vType):
  if vType == "xen":
    conn = libvirt.open('xen://')
  else:
    conn = libvirt.open('qemu:///system')
  vm = conn.lookupByName(vmName)

  #xmlFile = open(torrentDir + '/' + vmName + '.xml', 'w')
  #xmlFile = open('/tmp/' + vmName + '.xml', 'w')
  newVmXml = prepareXml(vm.XMLDesc(libvirt.VIR_DOMAIN_XML_SECURE)
      )
  #save a temp copy, perhaps for debugging :)
  #newVmXml.writexml(xmlFile)
  #define the new VM in libvirt
  vm = conn.defineXML(newVmXml.toxml())
  if autostart:
```

```
73      vm.create()
74    #xmlFile.close()
75
76  def cloneHdd(hdd, newHddPath):
77    cloneMethod = {
78      'kvm'  : cloneHddKvm,
79      'xen'  : cloneHddXen,
80      'qemu' : cloneHddQemu}
81    cloneMethod[config.vType](hdd, newHddPath)
82
83  def cloneHddQemu(hdd, newHddPath):
84    command = ['qemu-img', 'info' , hdd]
85    baseFormat = re.search('file␣format:␣(?P<format>[\S]*)',
          execute(command)).groupdict()['format']
86    command = ['qemu-img', 'create', '-f', 'qcow2', '-b', hdd, '-o'
          ,'backing_fmt=' + baseFormat, newHddPath]
87    execute(command)
88
89  def cloneHddKvm(hdd, newHddPath):
90    command = ['kvm-img', 'info' , hdd]
91    baseFormat = re.search('file␣format:␣(?P<format>[\S]*)',
          execute(command)).groupdict()['format']
92    command = ['kvm-img', 'create', '-f', 'qcow2', '-b', hdd, '-o',
          'backing_fmt=' + baseFormat, newHddPath]
93    execute(command)
94
95  def cloneHddXen(hdd, newHddPath):
96    command = ['vhd-util', 'snapshot', '-n', newHddPath, '-p', hdd]
97    execute(command)
98
99  if len(sys.argv) == 5:
100   startTimeSkript = time.time()
101   configPickle = open(os.path.expanduser('~/.config/whoami.pickle
          '), 'r')
102   config = pickle.load(configPickle)
103   configPickle.close()
104
105   vmName = sys.argv[1]
106   cloneCount = int(sys.argv[2])
107   if sys.argv[3] == 'y' or sys.argv[3] == 'Y' or sys.argv[3] == '
          j' or sys.argv[3] == 'J':
108     autostart = 'y'
109   else:
110     autostart = None
111   debug = int(sys.argv[4])
```

```
112
113   for i in range (0,cloneCount):
114     startTimeClone = time.time()
115     newVmName = randomName(vmName)
116     cloneVm(vmName, config.vType)
117     print 'Klon ' + str(i) + ': ' + newVmName
118     debugOut('Time needed: ' + str(time.time() - startTimeClone)
            , 1)
119   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=
         subprocess.PIPE, stderr=subprocess.PIPE)
19   stdout, stderr = s.communicate()
20   if debug > 0:
21     print stdout
22     print stderr
23   return stdout
24
25 def hddList(xmlDescription):
26   hList = []
27   description = parseString(xmlDescription)
28   hardDisks = description.getElementsByTagName("disk")
29   for i in range(0,len(hardDisks)):
30     if hardDisks[i].getAttribute("device") == "disk":
31       hardDisk = hardDisks[i].getElementsByTagName("source")[0].
             getAttribute("file")
```

```
32        hList.append(hardDisk)
33        """hardDiskFile = hardDisks[i].getElementsByTagName("source
              ")[0].getAttribute("file")
34        hardDiskDev = hardDisks[i].getElementsByTagName("source")
              [0].getAttribute("dev")
35        if hardDiskFile:
36          hList.append(hardDiskFile)
37        elif hardDiskDev:
38          hList.append(hardDiskDev)"""
39    return hList
40
41  def prepareXml(xmlDescription, vmName):
42    hList = []
43    description = parseString(xmlDescription)
44    hardDisks = description.getElementsByTagName("disk")
45    for i in range(0,len(hardDisks)):
46      if hardDisks[i].getAttribute("device") == "disk":
47        hardDisk = hardDisks[i].getElementsByTagName("source")[0].
              getAttribute("file")
48        hardDisks[i].getElementsByTagName("source")[0].setAttribute
              ("file", imageDir + "/" + vmName + "/" + os.path.
              basename(hardDisk))
49    #remove uuid and mac address
50    removeTag = description.getElementsByTagName("mac")[0]
51    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(modifedHddList(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:
          nodesll' + 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')
        , 'r')
98  config = pickle.load(configPickle)
99  configPickle.close()
100 vmName = sys.argv[1]
101 downloadDir = config.downloadDir
102 imageDir = config.imageDir
103 vType = config.vType
104 debug = 1
105 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=
       subprocess.PIPE, stderr=subprocess.PIPE)
10   stdout, stderr = s.communicate()
11   print stdout
12   print stderr
13   return stdout
14
15 packages = re.split('[\s]+', sys.argv[1])
16
17 command = ['apt-get', 'install', '-y'] + packages
18 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=
       subprocess.PIPE)
22   #stdout, stderr = s.communicate()
23   #if debug > 0:
24   #print stdout
```

```
25    #print stderr
26    return s.communicate()
27
28  name = gethostname()
29  ip = sys.argv[1]
30  command = ['find', "/boot/","-name","xen*.gz"]
31  stdout = execute(command)
32
33  if stdout and re.match('[\S]*xen[\S]*',os.uname()[2]):
34    vType = "xen"
35  else:
36    vType = "kvm"
37
38  downloadDir = sys.argv[2]
39  imageDir = sys.argv[3]
40
41  obj = VHost(ip, name, vType, imageDir, downloadDir)
42  if not os.path.exists(os.path.expanduser('~/.config')):
43    os.mkdir(os.path.expanduser('~/.config'))
44  cowpickle = open(os.path.expanduser('~/.config/whoami.pickle'), '
      w')
45  pickle.dump(obj, cowpickle)
46  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')
      , 'r').readlines()
14
15  def httpNo():
16    for line in configLines:
17      if re.match('[\s]*\(xend-http-server␣no\)[\s]*',line):
18        return
19
```

```
20    config = open('/etc/xen/xend-config.sxp','a')
21    config.write('\n(xend-http-server no)\n')
22    config.close()
23
24  def unixYes():
25    for line in configLines:
26      if re.match('[\s]*\(xend-unix-server yes\)[\s]*',line):
27        return
28
29    config = open('/etc/xen/xend-config.sxp','a')
30    config.write('\n(xend-unix-server yes)\n')
31    config.close()
32
33  httpNo()
34  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/OYKiAw, 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

[lib] redhat: *Setting up libvirt for TLS*. http://wiki.libvirt.org/page/TLSSetup, Abruf: 10.01.2011

[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.mspx, 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

Listings