# T U M

I N S T I T U T F Ü R I N F O R M A T I K

# Hochverfügbarkeit für Linux

Franz Huber, Tobias Schröpf

TUM-I0828
August 08

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-08-I0828-0/1.-FI


Druck: Institut für Informatik der
Technischen Universität München

# Hochverfügbarkeit für Linux

Franz Huber, Tobias Schröpf

Institut für Informatik,
Technische Universität München

{huberf,schroepf}@in.tum.de

## Zusammenfassung

Mit dem zunehmenden Einsatz von Linux-basierten Serversystemen auch im Rechenzentrumsbetrieb hat dieses frei verfügbare Betriebssystem ein Anwendungsfeld erreicht, in dem die permanente Verfügbarkeit der bereitgestellten Dienste eine wesentliche Anforderung ist. Sogenannte Hochverfügbarkeitslösungen sind in geschäftskritischen Einsatzbereichen daher auch für Linux nicht mehr wegzudenken.

Hochverfügbare Systeme sind dadurch gekennzeichnet, dass möglichst alle Fehlerquellen, die einen Systemausfall herbeiführen können, eliminiert werden. Das Beseitigen dieser „Single Points of Failure“ erfolgt in erster Linie durch Duplizierung der Systeme.

Diese Arbeit gibt einen Überblick über den aktuellen Stand der Hochverfügbarkeitslösungen für Linux und befasst sich daher hauptsächlich mit freien, OpenSource Hochverfügbarkeitslösungen. Als Evaluationsplattform dient eine mit dem freien Hypervisor Xen virtualisierte Server-Infrastruktur. Hier werden Mechanismen untersucht, um Produktivsysteme durch passive Standby-Systeme ausfallsicher zu machen. Ein weiterer Ansatz, der ebenfalls untersucht wird, sind mehrere aktive Systeme, die im Normalbetrieb die Anfragen untereinander aufteilen, und bei Ausfällen die Anfragen der ausgefallenen Systeme bearbeiten.

# Inhaltsverzeichnis

**Inhaltsverzeichnis** II

**1 Einführung** 1

**2 Lösungsansätze** 3

2.1 Grundlegende Verfahren . . . 3

2.1.1 Gemeinsamer Zugriff auf Daten und Datenreplikation . . . 3

2.1.2 Kanalredundanz . . . 3

2.1.3 Failoverlösungen . . . 4

2.1.4 Loadbalancing . . . 4

2.2 Kommerzielle HA-Lösungen . . . 4

2.2.1 Microsoft Windows Server 2003 . . . 5

2.2.2 Kommerzielle Linux/Unix Lösungen . . . 6

2.2.3 Hochverfügbarkeit in VMware-Umgebungen . . . 7

2.2.4 Kanalredundanz . . . 8

2.3 Open-Source Projekte . . . 9

2.3.1 Datenreplikation . . . 9

2.3.2 Shared Storage mit Netzwerk- und Cluster-Dateisystemen . . . 10

2.3.3 Kanal-Redundanz durch Channel-Bonding . . . 10

2.3.4 Failover mit Heartbeat . . . 11

2.3.5 Loadbalancing Lösungen . . . 11

2.3.6 Hochverfügbarkeit in virtualisierten Umgebungen . . . 12

**3 Zielplattform** 14

3.1 Architektur des Server-Pools . . . 14

3.2 Einsatzszenario und Single Points of Failure . . . 16

**4 Bewertung und Auswahl** 18

**5 Umsetzung** **20**

5.1 Konventionen für die Beispiele . . . . . 20

5.2 Failover mit Heartbeat . . . . . 20

5.2.1 Funktionsweise . . . . . 21

5.2.2 Architektur von Heartbeat . . . . . 21

5.2.3 Installation und Konfiguration . . . . . 24

5.2.4 Verwaltung und Überwachung des Clusters . . . . . 25

5.2.5 Resource-Agents . . . . . 27

5.3 Loadbalancing mit Linux Virtual Server . . . . . 30

5.3.1 Notwendige Anpassungen der Firewalls . . . . . 32

5.3.2 Konfiguration für LVS/NAT . . . . . 33

5.3.3 Konfiguration für LVS/DR . . . . . 35

5.3.4 Konfiguration für LVS/TUN . . . . . 36

5.3.5 Bewertung der Methoden zur Weiterleitung . . . . . 37

5.3.6 Failover für LVS mit Keepalived . . . . . 38

5.3.7 Installation von Keepalived . . . . . 38

5.3.8 Konfiguration von Keepalived . . . . . 38

5.3.9 LVS-Setup für Mail- und Webserver . . . . . 41

5.4 Loadsharing mit ClusterIP . . . . . 42

5.4.1 Installation und Konfiguration . . . . . 42

5.4.2 Failover . . . . . 43

**6 Zusammenfassung und Ausblick** **45**

**A Wichtige Begriffe** **47**

**B Konfigurationsdateien** **49**

B.1 Grundkonfiguration von Heartbeat . . . . . 49

B.1.1 /etc/ha.d/ha.cf . . . . . 49

B.1.2 /etc/ha.d/ha_logd.cf . . . . . 50

B.1.3 /etc/ha.d/authkeys . . . . . 50

B.2 IPaddr Resource Agent . . . . . 50

B.3 Gentoo Service Resource-Agent . . . . . 51

B.4 Filesystem Resource-Agent . . . . . 54

B.5 Beispiel für die CIB eines Mail-Servers . . . . . 55
B.6 Beispiel für die CIB eines NFS-Servers . . . . . 57
B.7 Anpassung der Firewall für LVS/NAT und LVS/DR . . . . . 58
B.8 Firewall-Skript für einen LVS/NAT-Director . . . . . 60
B.9 Netzwerkkonfiguration der Real Server für LVS/DR . . . . . 61
B.10 Netzwerkkonfiguration der Real Server für LVS/TUN . . . . . 62
B.11 Konfiguration eines Keepalived-Clusters . . . . . 62
B.12 Heartbeat Ressource für ClusterIP . . . . . 68

**Literaturverzeichnis** **70**

# 1 Einführung

Die Abhängigkeit von Diensten und Daten, die durch informationstechnische Systeme bereitgestellt werden, nimmt in allen Arbeits- und Lebensbereichen ständig zu. Um eine möglichst hohe Verfügbarkeit dieser Dienste und Daten zu erreichen, d.h. um die Erreichbarkeit der Dienste und den Zugriff auf die Daten möglichst andauernd zu gewährleisten, werden in der IT Hochverfügbarkeitssysteme entwickelt und immer weiter verbessert. In so genannten Hochverfügbarkeitsclustern (engl. High-Availability-Cluster, kurz HA-Cluster) werden einzelne Rechner (die Clusterknoten), die alle die gleiche Aufgabe erfüllen können, zu einem Cluster zusammengefasst, in dem meistens nur ein Knoten aktiv ist. Alle anderen Knoten sind passiv und warten darauf, den aktiven Knoten zu ersetzen, falls dieser ausfällt. In diesem Fall spricht man von einem *Failover*. Die Ausdehnung und Komplexität solcher HA-Cluster ist praktisch nicht begrenzt. So gibt es auch Cluster, die sich über hunderte Kilometer oder sogar über Kontinente hinweg erstrecken, um auch bei Naturkatastrophen an einem Cluster-Standort immer noch auf ein Backup-System umschalten zu können, das von dieser Katastrophe nicht betroffen ist. Allen Hochverfügbarkeitslösungen ist gemeinsam, dass sie versuchen einzelne Komponenten, deren Ausfall einen Ausfall des Gesamtsystems nach sich ziehen würde (sog. *Single Points of Failure*, kurz *SPOF*), ausfallsicher zu machen.

Generell versteht man unter der Verfügbarkeit eines IT-Systems das Verhältnis aus dem Zeitraum, während dessen das System die ihm zugedachten Aufgaben (Dienste) erfüllen kann (*Uptime*), zum Gesamtzeitraum, bestehend aus der Uptime und den Phasen, in denen das System seine Aufgaben aufgrund verschiedenster Ursachen nicht erfüllen kann (*Downtime*). Downtimes können dabei ungeplant sein, z.B. aufgrund von Hardwareausfällen, oder geplant, im Rahmen von Wartungsphasen der Hard- und/oder Software des Systems. Die Verfügbarkeit eines IT-Systems wird häufig in Prozent angegeben und nach verschiedenen Ansätzen in sogenannte Verfügbarkeitsklassen eingeteilt. Gebräuchlich ist z.B. die Einteilung in mit Zahlen bezeichnete Klassen, entsprechend der Anzahl der Ziffer 9 in der prozentualen Verfügbarkeitsangabe. So entspricht die Klasse 5 (99,999% Verfügbarkeit) einer Downtime von maximal 26,3 Sekunden pro Monat oder 5,26 Minuten pro Jahr. Für ein IT-System, das Verfügbarkeitsanforderungen der Klasse 5 zu erfüllen hat, ist mithin der Einsatz von HA-Techniken unumgänglich, allein schon aufgrund der Tatsache, dass sich der für Wartungsarbeiten erforderliche Zeitraum im Jahr nur in den seltensten Fällen auf den geforderten Rahmen von ca. 5 Minuten beschränken wird.

Das freie Betriebssystem Linux findet zunehmend auch in Bereichen Anwendung, die noch vor Jahren der klassischen „Rechenzentrums-IT“ zugedacht waren und einerseits durch Mainframes, andererseits durch Serversysteme mit etablierten, kom-

merziellen UNIX-Betriebssystemen wie AIX, HP-UX oder Solaris bedient wurden. Konsequenterweise bewegen sich damit auch die Verfügbarkeitsanforderungen an Linux-Systeme, welche in unternehmenskritischen Anwendungen eingesetzt werden, in ähnlichen Bereichen wie für Mainframes gewohnt und machen den Einsatz von HA-Lösungen zwingend erforderlich.

Ziel dieser Arbeit ist es, im Hinblick auf einen möglichen produktiven Einsatz von Linux-basierten HA-Technologien in der IT am Lehrstuhl für Software & Systems Engineering der Fakultät für Informatik an der TU München den aktuellen Stand der verfügbaren HA-Lösungen für Linux darzustellen und geeignete Ansätze zur detaillierten Evaluierung auf Basis der existierenden IT-Infrastruktur operational umzusetzen. Bei dieser IT-Infrastruktur handelt es sich um einen Pool von x86_64 Servern, auf denen eine mithilfe des freien Hypervisors *Xen* virtualisierte *Gentoo* Linux-Plattform betrieben wird. Beispielanwendung für die Umsetzung ist ein aus drei Subsystemen bestehendes Mailsystem.

Die nachfolgenden Abschnitte der vorliegenden Arbeit gliedern sich wie folgt: Wir geben in Abschnitt 2 zunächst einen Überblick über gängige Ansätze Systeme hochverfügbar zu machen. Neben den prinzipiellen Herangehensweisen betrachten wir dort im Sinne eines Marktüberblicks sowohl kommerzielle als auch Open-Source HA-Lösungen – nicht ausschließlich für Linux. Im anschließenden Abschnitt 3 wird kurz die der Evaluierung als Basis dienende Infrastruktur beschrieben. Abschnitt 4 bewertet die aufgeführten HA-Lösungen und gibt eine Motivation für die drei für eine detailliertere Evaluation ausgewählten Lösungsansätze. Die konkrete Implementierung der drei HA-Ansätze – Heartbeat, Linux Virtual Server und ClusterIP – auf der Evaluierungsplattform wird in Abschnitt 5, dem Hauptteil der Arbeit, im Einzelnen dargestellt. Die Arbeit endet mit einer kurzen Zusammenfassung und einer abschließenden Bewertung des Einsatzes von HA-Lösungen. Der Anhang der Arbeit umfasst für alle drei umgesetzten HA-Lösungen die erforderlichen Konfigurationsdateien sowie, falls erforderlich, notwendige Patches.

# 2 Lösungsansätze

Um in einem IT-System alle denkbaren Single Points of Failure zu eliminieren, muss an verschiedenen Stellen im Systemaufbau angesetzt werden. So ist neben der offensichtlichen Ausfallsicherheit der eigentlichen Serversysteme u.a. auch dafür Sorge zu tragen, dass die Datenhaltung ausfallsicher angelegt ist und dass die Kommunikation der Systeme untereinander sowie mit der Umwelt auch z.B. bei Unterbrechung einzelner Netzwerkverbindungen weiter gewährleistet ist. Wir charakterisieren zunächst kurz grundlegende Herangehensweisen, die Ausfallsicherheit eines IT-Systems zu erreichen, und gehen dann auf kommerzielle sowie OpenSource-Lösungen ein, die einzelne oder mehrere Aspekte der Hochverfügbarkeit abdecken.

## 2.1 Grundlegende Verfahren

### 2.1.1 Gemeinsamer Zugriff auf Daten und Datenreplikation

Die Notwendigkeit von Backups der Nutzerdaten ist unbestritten. Bei Datenverlusten jeglicher Art muss eine Möglichkeit existieren, verlorene Daten zu restaurieren. In gewisser Weise erhöhen auch regelmäßige Backups bereits die Verfügbarkeit eines Systems, da bei Datenverlusten die verlorenen Daten aus dem Backup restauriert werden können. Da das Restaurieren der Daten meist von Hand ausgeführt werden muss und in vielen Fällen sehr zeitaufwändig sein kann, eignen sich herkömmliche Backups nicht als Grundlage für hochverfügbare Systeme.

Für Failoversysteme reicht das Erstellen regelmäßiger Backups der Daten nicht aus, da alle Knoten eines HA-Clusters Zugriff auf den gleichen Datenbestand benötigen. Hier ist es notwendig, die Daten zwischen den Knoten zu replizieren, oder ihnen die Daten auf einem gemeinsamen Speichermedium (*Shared Storage*), bei konkurrierendem Zugriff auf die Daten unter Verwendung eines Clusterdateisystems, zur Verfügung zu stellen. Kommt ein *Shared Storage* zum Einsatz, sollte auch dieses in regelmäßigen, möglichst kurzen Zeitabständen repliziert werden – idealerweise durch die direkte Spiegelung zwischen mehreren Speichersystemen auf Hardwareebene –, um bei einem Ausfall dieses Systems möglichst transparent auf ein Ersatzsystem umschalten zu können.

### 2.1.2 Kanalredundanz

Für Serverdienste ist es wichtig, dass sie möglichst ohne Unterbrechungen erreichbar sind. Für die Kommunikationssysteme bedeutet dies, dass sie hochverfügbar sein

müssen. Ein Weg, die Verfügbarkeit der Kommunikationssysteme zu erhöhen, ist es, die Kommunikationskanäle redundant auszulegen, um beim Ausfall eines Kanals immer noch alternative Kommunikationskanäle zur Verfügung zu haben.

### 2.1.3 Failoverlösungen

Failoverlösungen zeichnen sich dadurch aus, dass dem Produktivsystem ein oder mehrere unabhängige Systeme zur Seite gestellt werden, die im Ernstfall alle Aufgaben des Produktivsystems übernehmen können. Das Umschalten vom Produktiv- zum Standby-System (also der *Failover*) sollte in einem solchen HA-Cluster möglichst schnell und transparent für die Anwender erfolgen.

Failoverlösungen kommen vor allem in *Aktiv/Passiv-Clustern* zum Einsatz, also in Clustern, in denen nur ein Knoten aktiv Dienste anbietet, während die restlichen Knoten passiv sind und erst aktiv werden, wenn der aktive Knoten ausfällt.

### 2.1.4 Loadbalancing

Ähnlich wie bei den Failoverlösungen ist auch ein Loadbalancing-Cluster aus mehreren Systemen aufgebaut, von denen jedes in der Lage ist, die Benutzeranfragen zu beantworten. Anders als bei Failoversystemen sind hier alle Systeme aktiv und beantworten abwechselnd die Anfragen. Hierbei handelt es sich um *Aktiv/Aktiv-Cluster*. Als wichtiger Bestandteil kommt bei Loadbalancing-Systemen in der Regel deshalb noch ein sogenannter Loadbalancer zum Einsatz, der alle Anfragen an das Gesamtsystem auf die einzelnen Knoten des Loadbalancing-Clusters verteilt.

Eine besondere Form des Loadbalancing wird in dieser Arbeit als *Loadsharing* bezeichnet. Die Loadsharing-Systeme kommen ohne Loadbalancer aus. Sie teilen sich die Anfragen gemäß einer festgelegten Policy auf.

Da Loadbalancing hauptsächlich auf die Performanz der Systeme abzielt und weniger als Hochverfügbarkeitslösung eingesetzt wird, muss darauf geachtet werden, dass Loadbalancing meist mit Failoversystemen gekoppelt werden muss, um tatsächlich die Verfügbarkeit des Systems zu erhöhen.

## 2.2 Kommerzielle HA-Lösungen

Sowohl die kommerziellen wie auch die freien Hochverfügbarkeitslösungen am Markt sind zum Großteil für den Einsatz auf Unix- bzw. Linux-Plattformen bestimmt. Manche Windows-Betriebssysteme unterstützen jedoch bereits mit Bordmitteln mehrere der genannten Lösungsansätze.

### 2.2.1 Microsoft Windows Server 2003

In Microsoft Windows Server 2003 sind bereits einige Technologien integriert, mit deren Hilfe sich drei der vier oben genannten Lösungsansätze verwirklichen lassen.

Das *Distributed File System* (kurz *DFS*) erlaubt es freigegebene Verzeichnisse verschiedener Fileserver zu sogenannten *DFS Namespaces* zu gruppieren. Die Adressierung von freigegebenen Verzeichnissen erfolgt bei DFS nicht durch explizites Angeben des Fileservers, der das Verzeichnis freigibt, sondern transparent über den *DFS Namespace*, zu dem das Verzeichnis gehört. Dadurch ist es möglich die gleiche DFS-Freigabe durch mehrere Server bereitstellen zu lassen. Der Zugriff erfolgt dann wieder transparent über den *DFS Namespace*. Fällt einer der Fileserver aus, so wird, für den Anwender transparent, über einen anderen Server auf die Dateien zugegriffen. Mit dieser Methode sind die Daten solange erreichbar, wie mindestens einer der Server der DFS-Freigabe funktionsfähig ist (siehe [4]). Der Datenbestand der DFS-Server muss jedoch mit zusätzlichen Hilfsmitteln synchronisiert werden. Für diese Aufgabe bietet das Windows Betriebssystem den *File Replication Service* (kurz *FRS*) an. Der *FRS* erkennt Änderungen innerhalb der replizierten Verzeichnisse und synchronisiert die Änderungen auf die anderen Server. Da *FRS multimasterfähig* ist, darf jeder Server, der an dem Replikationsdienst teilnimmt, Änderungen an den Daten vornehmen und diese dann an die restlichen Server replizieren (siehe [5]). Mit einer Kombination dieser beiden Technologien ist es also möglich, den Datenbestand konsistent auf mehrere Server zu verteilen und dadurch ein hochverfügbares Shared Storage aufzubauen.

Mit den *Server Clustern* stellt der *Windows Server 2003* außerdem eine Technologie bereit, mit der sich Dienste und Anwendungen mit Hilfe einer Failoverlösung hochverfügbar machen lassen. Ein *Server Cluster* darf aus bis zu acht Knoten bestehen, von denen alle auf einen gemeinsamen Datenspeicher zugreifen können. Alle Anwendungsdaten, die für ein Failover benötigt werden, werden auf diesem gemeinsamen Datenspeicher abgelegt. Ein *Failover* im *Server Cluster* erfolgt transparent für die Anwender. Die Ressourcen, die bei einem Failover übernommen werden, werden in Gruppen spezifiziert. Wenn ein Server ausfällt wird die komplette Ressourcengruppe auf einen anderen Server migriert. Außerdem ist es möglich den Cluster für *Failback* zu konfigurieren, um die ausgelagerten Ressourcen auf den ursprünglichen Server zurück zu migrieren, sobald dieser wieder funktionsfähig ist (siehe [17]).

Neben Failover-Lösungen lassen sich mit dem Windows Server 2003 auch Loadbalancing-Lösungen verwirklichen. Die Technologie, mit der sich ein Loadbalancing-Cluster in Windows umsetzen lässt, heißt *Network Load Balancing* (kurz *NLB*). Ein NLB-Cluster darf aus bis zu 32 Knoten bestehen, die im Gegensatz zu den *Server Clustern* keinen gemeinsamen Cluster-Speicher verwenden, sondern alle Anwendungen und Daten lokal speichern. Alle Client-Anfragen werden an eine virtuelle IP Adresse geschickt, die dem NLB-Cluster zugeordnet ist. Alle Pakete an diese Adresse werden von allen Knoten empfangen, welche dann anhand eines Hashing-Verfahrens entscheiden, welcher der Knoten die Anfrage annimmt und bearbeitet. Durch den Ansatz mit der virtuellen IP Adresse ist ein transparentes *Failover* sowie *Failback* möglich, da alle Knoten mit Hilfe von *Heartbeats* ständig die Erreichbar-

keit der anderen Cluster-Knoten überprüfen und nicht erreichbare Knoten aus dem Cluster entfernen, beziehungsweise wieder in den Cluster integrieren, sobald sie wieder erreichbar sind (siehe [15]). Die Synchronisation der lokalen Datenbestände aller Knoten kann dann beispielsweise mit Hilfe des oben genannten *FRS* erfolgen.

Da das NLB von Windows ohne Loadbalancer auskommt handelt es sich um einen Vertreter der Loadsharing-Technologien.

Der *Windows Server 2003* bietet also einige Möglichkeiten an, um die Verfügbarkeit von Systemen zu erhöhen. Mit *DFS* und *FRS* steht ein Weg zur Verfügung, um Daten auf mehrere Speichersysteme zu synchronisieren und den Zugriff auf diese Daten ausfallsicher und transparent zu ermöglichen. Mit *Server Clustern* können Hochverfügbarkeitscluster aufgebaut werden, die automatisches *Failover* unterstützen. Mit *NLB* können hingegen lastverteilende Cluster erstellt werden, die gleichzeitig auch automatisches *Failover* ermöglichen.

### 2.2.2 Kommerzielle Linux/Unix Lösungen

#### IBM-Produkte

IBM bietet für seine Plattformen zahlreiche Produkte im Bereich Hochverfügbarkeit an.

Mit dem *General Parallel File System* (*GPFS*) steht ein Cluster-Dateisystem zur Verfügung, das es ermöglicht mehr als 1000 Festplatten zu einem Dateisystem mit vielen Terabytes zusammenzufassen. Jeder Knoten, der das *GPFS* gemountet hat, darf zu jeder Zeit lesend und schreibend auf das Dateisystem zugreifen. Die einzelnen Dateien werden in Blöcke unterteilt und über mehrere Festplatten verteilt. Somit ist es möglich, auf unterschiedliche Teile einer Datei gleichzeitig von verschiedenen Knoten aus zuzugreifen. Durch Replikations- und Failovermechanismen können *Single Points of Failure* ausgeschaltet werden, was die Verfügbarkeit der Daten erhöht (siehe [7]).

Die Failover-Lösung von IBM heißt *High Availability Cluster Multiprocessing* (kurz *HACMP*). Ursprünglich wurde sie für das UNIX Betriebssystem AIX entwickelt. Mittlerweile läuft die Software mit Einschränkungen auch unter Linux auf IBM Hardware. Neben den üblichen Mechanismen zur Überwachung und Fehlererkennung steht mit HACMP/XD eine Erweiterung zur Verfügung, die die Daten über weite Entfernungen spiegeln und synchronisieren, sowie einen *Failover* durchführen kann.

#### Sun Cluster

Die Clustersoftware *Sun Cluster* der Firma Sun bietet Failover-Mechanismen für Solaris Plattformen. Die Cluster-Knoten sind über ein privates Netzwerk miteinander verbunden. Über dieses Netzwerk werden sowohl Daten als auch *Heartbeats* verschickt. Mit Hilfe der *Heartbeats* können die Knoten Ausfälle anderer Knoten erkennen und darauf reagieren. So wird ein ausgefallener Knoten im Cluster isoliert

und die restlichen Knoten übernehmen dessen Dienste und Daten. Dieser *Failover* erfolgt für die Clients transparent.

#### HP ServiceGuard

Hewlett-Packard bietet mit ServiceGuard eine ähnliche Software an. Auf Basis eines gemeinsamen Speichers wird sichergestellt, dass alle Knoten immer Zugriff auf den aktuellsten Datenbestand haben. Dienste und Ressourcen werden hier zu Paketen gruppiert. Mit Hilfe von Heartbeats, die alle Knoten regelmäßig emittieren, erkennen die Knoten die Ausfälle anderer Knoten. In solch einem Fall werden alle Dienste des ausgefallenen Knotens auf einen anderen Knoten im Cluster transferiert.

#### Symantec

Symantec bietet die Veritas-Produktreihe für verschiedene Betriebssysteme an, wodurch sich die Flexibilität gegenüber anderen HA-Lösungen deutlich erhöht. Der *Veritas Volume Replicator* ermöglicht, ähnlich wie HACMP von IBM, die Synchronisation von Daten und Datenbanken über beliebige Distanzen hinweg. Außerdem bietet er die Möglichkeit, mit Hilfe von Bandbreitenmodellierung, die Synchronisation an die verfügbare Bandbreite anzupassen (siehe [19]).

Wie andere HA-Lösungen überwacht auch der *Veritas Cluster Server* die Erreichbarkeit und Funktionsfähigkeit von Anwendungen und führt im Fehlerfall einen Failover zum Sekundärsystem durch. Der *Veritas Cluster Server* zeichnet sich vor allem dadurch aus, dass er anhand von definierbaren Regeln entscheidet, welches Sekundärsystem am besten für den Failover geeignet ist. Dadurch, dass sowohl UNIX (Solaris, AIX, HP-UX), Linux (Red Hat, SUSE) wie auch Windows unterstützt werden, lässt sich diese Software vielseitig, und in den meisten Umgebungen einsetzen.

#### SteelEye LifeKeeper

Ein weiteres kommerzielles HA-Produkt für Linux ist *LifeKeeper* von der Firma *SteelEye*. Ähnlich wie die vorangegangenen Produkte überwacht auch diese Software geclusterte Linux Systeme. Falls ein Server im Cluster ausfällt, übernehmen andere Server (für den Anwender transparent) dessen Dienste. Außerdem ist es möglich, bei Bedarf weitere Knoten zum Cluster hinzuzufügen. Die Daten liegen bei diesem System auf einem gemeinsamen zentralen Speicher, so dass alle Server unabhängig von den Daten sind und jeder Server auf die Daten zugreifen kann, die er benötigt. Die Integrität der Daten wird durch Sperrmechanismen gewährleistet, die es einzelnen Anwendungen im Cluster ermöglichen exklusiv auf die jeweiligen Daten zuzugreifen.

### 2.2.3 Hochverfügbarkeit in VMware-Umgebungen

Die Produktpalette von VMware umfasst mehrere interessante Technologien, mit denen sich die Vorteile von Virtualisierungslösungen für die Hochverfügbarkeit nutzen

lassen. VMware HA (siehe [22]) erkennt Ausfälle von physischen Servern und startet alle virtuellen Maschinen, die vor dem Ausfall darauf liefen, auf anderen physischen Servern neu. Dabei ist es möglich, mit VMware DRS (siehe [21]), den physischen Server, auf dem die virtuellen Maschinen gestartet werden sollen, anhand der zur Verfügung stehenden Ressourcen, dynamisch zu wählen.

Auch die Live-Migration von virtuellen Maschinen ist möglich. Mit VMware VMotion (siehe [23]) können virtuelle Maschinen im laufenden Betrieb auf andere physische Server verschoben werden. Diese Funktion bietet sich an, wenn Wartungsarbeiten an der Server-Hardware oder dem Betriebssystem des physischen Servers notwendig sind.

### 2.2.4 Kanalredundanz

Alle bisher vorgestellten Lösungen beschränken sich auf die Bereiche Datenreplikation, Failover und Loadbalancing. In diesem Abschnitt werden Möglichkeiten vorgestellt, um auch die Kommunikationskanäle als SPOFs zu eliminieren.

#### HP Teaming

Durch die Bündelung mehrerer Netzwerkverbindungen zu einer einzigen Verbindung können beim *HP Teaming* gleichzeitig Datendurchsatz und Verfügbarkeit der Netzwerkverbindung erhöht werden. Ein *Team* kann aus zwei bis acht Ports bestehen. Es kann wie eine einzige virtuelle *NIC* verwendet werden. Wenn eine der einzelnen Verbindungen ausfällt, läuft die Kommunikation über die restlichen Kanäle. Außerdem stehen verschiedene Loadbalancing Strategien zur Verfügung, um den Datenverkehr auf die Kanäle eines Teams zu verteilen (siehe [8]).

#### Cisco EtherChannel

Bei *Cisco EtherChannel* handelt es sich um eine ähnliche Technologie. Auch hier können bis zu acht herkömmliche Ethernet-Verbindungen zu einer einzigen Verbindung zusammengefasst werden. Wie auch beim HP Teaming, wird im Normalbetrieb der Datenverkehr auf alle Kanäle verteilt. Fällt eine Verbindung aus, so läuft der Datenverkehr über die verbleibenden Kanäle. Es entsteht also eine fehlertolerante Datenverbindung. Die EtherChannel Technologie kann dabei für die Vernetzung von LAN Switches, Routern, Servern und Clients verwendet werden. Die Algorithmen, nach denen das Loadbalancing auf die einzelnen Kanäle durchgeführt wird, können abhängig von MAC Adresse, IP Adresse oder TCP/UDP Port definiert werden (siehe [2]).

## 2.3 Open-Source Projekte

### 2.3.1 Datenreplikation

Bei *Failover*-Lösungen ist es wichtig, dass das Produktivsystem und die Standby-Systeme Zugriff auf einen einheitlichen Datenbestand haben. Eine Möglichkeit hierfür stellt ein gemeinsames Speichermedium (*Shared Storage*) dar, das alle Cluster-Knoten bei Bedarf einmounten und verwenden können.

Ein anderer Ansatz, um alle Knoten mit dem aktuellen Datenbestand zu versorgen, sind sogenannte Netzwerk-Blockdevices. Diese ermöglichen den Zugriff auf Blockdevices über ein Netzwerk.

#### Netzwerk-Blockdevices

Für Failover-Szenarien oder Loadbalancing-Lösungen ist eine manuelle Spiegelung der Daten in regelmäßigen Zeitabständen nicht ausreichend. Kommt es kurz vor dem nächsten Backup-Zyklus zu einem Ausfall und folglich zum Failover auf ein Standby-System, so fallen alle Dienste des Systems zurück auf den Datenstand zur letzten Sicherung. Alle Änderungen seit dieser letzten Sicherung wären dann verloren.

Netzwerk-Blockdevices helfen, dieses Problem zu lösen. Die grundlegende Idee besteht darin, alle Änderungen am Dateisystem in Echtzeit über ein Netzwerk an ein Backup-System zu replizieren. Für ein eventuelles *Failover* liegen dann stets alle Daten in der aktuellen Version vor.

Die Datenreplikation kann beispielsweise über das *Network Block Device* (kurz *NBD*) erfolgen. Mit diesem Treiber, der in den Linux Kernel integriert ist, ist es dem Produktivsystem möglich, vom Standby-Knoten ein Blockdevice über eine Netzwerkverbindung einzumounten. Danach kann das Produktivsystem das Blockdevice wie eine herkömmliche Festplattenpartition verwenden und lesend wie schreibend darauf zugreifen. Die Replikation selbst kann dann mit Hilfe eines Software-RAID-Systems (Spiegelung, RAID 1) erfolgen.

Das *Distributed Replicated Block Device* (kurz *DRBD*) ist ebenfalls eine Software, die ein Blockdevice implementiert, das über das Netzwerk gemountet werden kann. Allerdings handelt es sich bei DRBD um eine Komplettlösung, die auch die Replikation ohne zusätzliche Hilfsmittel wie Software-RAIDs ermöglicht. Wird beispielsweise auf dem Produktivserver auf das DRBD-Device geschrieben, so werden die Daten einerseits direkt auf das darunterliegende lokale Blockdevice durchgeschrieben und gleichzeitig zum Sekundär-Device, d.h. über das Netzwerk zum Standby-Knoten, übermittelt und auf dessen Festplatten gespeichert.

#### Datenbankreplikation

Viele Datenbank-Server bieten Replikations-Mechanismen, die es erlauben alle Änderungen an der Originaldatenbank (Master) an einen oder mehrere Slaves zu sen-

den und somit eine funktionsfähige Kopie der Datenbank zu erhalten. MySQL unterstützt zwei verschiedene Konzepte zur Datenreplikation. Beim *synchronen* Ansatz (MySQL Cluster, siehe [13]) verwenden die gleichberechtigten MySQL-Knoten eine sogenannte *NDB Cluster storage engine*, die den Cluster hochverfügbar macht und die Daten konsistent hält. Beim *asynchronen* Ansatz (MySQL Replication, siehe [14]) speichert das Produktivsystem alle Transaktionen in einem sogenannten Binary-Log, mit dessen Hilfe sich die Slaves regelmäßig synchronisieren können.

### 2.3.2 Shared Storage mit Netzwerk- und Cluster-Dateisystemen

Für Failoverlösungen stellt ein gemeinsam genutzter Datenspeicher, also ein *Shared Storage*, eine Alternative zur Datenreplikation dar. Alle Clusterknoten können dieses Storage bei Bedarf einmounten und haben somit im Falle eines Failovers Zugriff auf den aktuellen Datenbestand.

Mit Hilfe von Netzwerk-Dateisystemen wie NFS (Network Filesystem) können Dateisysteme für mehrere Computer in einem Netzwerk freigegeben werden. So ist es möglich, dass mehrere Computer über ein Netzwerk auf das gleiche Dateisystem lesend wie schreibend zugreifen können. Über das Protokoll wird sichergestellt, dass konkurrierende Zugriffe auf das Dateisystem unterbunden bzw. in konsistenter Art und Weise abgearbeitet werden.

Im Gegensatz dazu benötigen Cluster-Dateisysteme keinen Server, der die Zugriffe auf das Dateisystem koordiniert. Alle Knoten im Cluster können gleichberechtigt und konkurrierend auf das Dateisystem zugreifen. Prominente Vertreter dieser Dateisysteme sind Red Hat's GFS und das OCFS2 von Oracle, das seit der Kernelversion 2.6.16 fester Bestandteil des Linux Kernels ist.

### 2.3.3 Kanal-Redundanz durch Channel-Bonding

Das *Linux Channel Bonding* (siehe [9]) ist bereits als Treiber im Linux Kernel enthalten und ermöglicht es dem Anwender mehrere Netzwerkschnittstellen zu einer logischen Schnittstelle zu bündeln.

Dieser Ansatz schützt das System vor Schicht 1 (Kabel) und Schicht 2 (Switch, Netzwerkkarte) Fehlern. Abhängig vom Modus, in dem der Treiber arbeitet, kann er ein *Loadbalancing* über alle physikalischen Netzwerkverbindungen ermöglichen (Balance Round Robin, Balance XOR, Balance TLB, Balance ALB), durch *Failover*-Mechanismen höhere Ausfallsicherheit bieten (Active Backup, Broadcast) oder beides (Balance Round Robin, Balance XOR). Die Ausfallsicherheit, und somit die Verfügbarkeit der Netzwerkverbindung, wird durch Channel Bonding erhöht, da beim Ausfall einer physikalischen Netzwerkverbindung auf funktionierende Netzwerkverbindungen ausgewichen werden kann. Somit sind defekte Netzwerkkarten, Netzwerkkabel oder Switches kein SPOF mehr (falls die physikalischen Verbindungen über mindestens zwei unabhängige Switches laufen).

### 2.3.4 Failover mit Heartbeat

Das Ziel des *High-Availability Linux* Projekts (siehe [18]) ist es, den Aufbau von Hochverfügbarkeitsclustern unter Linux, aber auch unter BSD- und Solaris-Systemen zu ermöglichen. Kernkomponente des Projekts ist eine Software namens *Heartbeat*. Sie stellt die nötige Funktionalität bereit, um das *Failover* von Diensten zu ermöglichen.

*Heartbeat* sendet über das Netzwerk Lebenszeichen (sog. *Heartbeats*) an die *Heartbeat*-Instanzen der anderen Clusterknoten. Empfängt ein Knoten keine *Heartbeat*-Nachrichten mehr, geht er davon aus, dass der andere Knoten offline ist und alle seine Dienste werden übernommen (*Failover*). Um sicher zu gehen, dass der andere Knoten wirklich offline ist, kann ein *STONITH*-Mechanismus („Shoot the other node in the head“) zum Einsatz kommen.

### 2.3.5 Loadbalancing Lösungen

#### Loadbalancing mit Linux Virtual Server

Das Open-Source Projekt *Linux Virtual Server* (siehe [10]) entwickelt eine Software für den Aufbau hoch skalierbarer Loadbalancing-Cluster unter Linux. Kernkomponente des Linux Virtual Server Projektes ist der Kerneltreiber IPVS (IP Virtual Server), der Loadbalancing-Mechanismen auf der Schicht 4 (Transportschicht) bereitstellt.

Ein mit Hilfe dieses Treibers implementierter Loadbalancer (auch *Director* genannt) verteilt dann (über *NATting*, *IP-Tunneling* oder *Direct Routing*) alle Client-Anfragen auf die verschiedenen Knoten (die *Real Server*) im LVS-Cluster. Auf dem *Director* können Monitoring-Tools eingesetzt werden, um Ausfälle und Fehlverhalten der *Real Server* zu erkennen. Wird ein Ausfall erkannt, streicht der Loadbalancer den Knoten aus der Liste der verfügbaren Server.

Durch diese Vorgehensweise entsteht jedoch ein neuer *Single Point of Failure*, nämlich der Loadbalancer. Fällt dieser aus, sind alle Dienste, die er vermittelt, nicht mehr erreichbar. Um dieses Szenario zu verhindern, kann dem Loadbalancer ein Backup-System zur Seite gestellt werden, das über eine *Heartbeat*-Verbindung Ausfälle des Loadbalancers erkennt und dessen Aufgaben übernimmt.

#### Keepalived für Failover im Linux Virtual Server Cluster

Die Software Keepalived stellt eine Ergänzung zu LVS dar, die einen LVS-Cluster um Failover-Mechanismen erweitern kann.

Ein LVS-Cluster muss dann aus mindestens zwei *Directors* und beliebig vielen *Real Servern* bestehen. In einer solchen Umgebung können dann die beiden Loadbalancer zum Einen Ausfälle des anderen Loadbalancers erkennen, und zum Anderen kann Keepalived so konfiguriert werden, dass es auch Ausfälle der Backends oder einzelner

Dienste der Backends erkennt, und entsprechend reagiert.

Zusammen mit LVS und Keepalived lässt sich somit ein hochverfügbarer Loadbalancing-Cluster verwirklichen, der sowohl mit Ausfällen von Loadbalancern, als auch mit Ausfällen der *Real Server* umgehen kann.

#### Crossroads

Bei Crossroads (siehe [3]) handelt es sich ebenfalls um eine Loadbalancing-Software. Anders als Linux Virtual Server arbeitet Crossroads allerdings im User-Space. Zusätzlich zu den üblichen Loadbalancing Funktionen kann diese Software erkennen, wenn Backends, also die Dienste an die die Anfragen weiter verteilt werden, ausfallen. In einem solchen Szenario werden alle Anfragen automatisch an die restlichen Backends verteilt. Wenn ein ausgefallener Dienst wieder erreichbar ist, wird er automatisch wieder in die Verteilung mit aufgenommen. Außerdem kann die Verteilung der Anfragen abhängig von der Last erfolgen.

Eine Gemeinsamkeit zu Linux Virtual Server ist das Entstehen eines neuen SPOFs am Loadbalancer. Die Maschine, auf der Crossroads läuft, muss mit Hilfe einer Failover-Lösung hochverfügbar gemacht werden, um eine größtmögliche Ausfallsicherheit zu gewährleisten.

#### ClusterIP

Eine andere Loadbalancing-Lösung, die ebenfalls fester Bestandteil neuerer Linux Kernel ist, heißt ClusterIP. ClusterIP ist Teil des Netfilter-Codes im Kernel. Die Funktionsweise ist allerdings unterschiedlich zu den beiden bereits vorgestellten Loadbalancing Ansätzen. Dieses System verwendet keinen separaten Loadbalancer. Stattdessen entscheiden die Server selbst, ob sie für eine Anfrage zuständig sind oder nicht. Somit wird auf elegante Weise der Loadbalancer als SPOF eliminiert. Allerdings muß den Cluster-Knoten zusätzlich ein Mechanismus bereitgestellt werden, mit dem sie Ausfälle anderer Knoten im Cluster erkennen.

Es erreichen alle Anfragen jeden Knoten im Cluster. Die Knoten entscheiden dann selbstständig anhand von Absenderadresse, Absenderport und Zielport ob sie diese Anfrage bearbeiten oder verwerfen. Bei ClusterIP handelt es sich demnach um eine freie Loadsharing-Lösung.

### 2.3.6 Hochverfügbarkeit in virtualisierten Umgebungen

#### Migration von U-Domänen in Xen

Wie auch andere Virtualisierungslösungen (vgl. Abschnitt 2.2.3) ist der freie Hypervisor *Xen* ([28]) zur Virtualisierung von (nicht nur) Linux-Instanzen sehr gut zum Einsatz innerhalb eines Hochverfügbarkeitsclusters geeignet.

Sind die Rootdateisysteme der virtualisierten Server (*U-Domänen*) auf einem zentralen, allen Xen-Managementeinheiten (*0-Domänen*) zugänglichen, Speichersystem abgelegt, so können sie auf beliebigen der verfügbaren 0-Domänen gestartet werden.

Fällt eine 0-Domäne (insbesondere: die zugehörige Serverhardware, auf der die 0-Domäne läuft) aus, so können alle U-Domänen, die zum Zeitpunkt des Ausfalls in dieser 0-Domäne liefen, auf einer anderen 0-Domäne neu gestartet werden. Mit Hilfe von Heartbeat kann dieses Verhalten automatisiert werden.

Zusätzlich ist es in Xen möglich, U-Domänen im laufenden Betrieb von einer 0-Domäne auf eine andere umzuziehen. Vor Wartungsarbeiten an der 0-Domäne oder der Hardware, auf der die 0-Domäne läuft, können alle U-Domänen der 0-Domäne in eine andere 0-Domäne verschoben werden (*Live-Migration*). Dabei kommt es zu einer maximalen *Downtime* von wenigen Sekunden.

### Verwaltung von Xen-U-Domänen mit Ganeti

Auch ein GPL-Projekt von Google befasst sich mit der Verwaltung von paravirtualisierten Systemen. Die Anwendung mit dem Namen Ganeti (siehe [6]) unterstützt Cluster die aus bis zu 25 Xen-U-Domänen bestehen und ermöglicht neben Starten und Herunterfahren der Domänen auch die Installation des Betriebssystems in den U-Domänen. Die für Hochverfügbarkeitsansätze wichtigste Funktion stellt jedoch die auf der Projektseite erwähnte *Failover*-Funktion dar, mit der ausgefallene U-Domänen auf andere 0-Domänen umgezogen werden können.

# 3 Zielplattform

Die Server-Infrastruktur am Lehrstuhl für Software & Systems Engineering stellt die technische Plattform für eine Vielzahl von unterschiedlichen IT-Services bereit, welche von Datei- über Mail- bis hin zu Web-Services reichen. Für einige dieser Dienste gelten hohe Anforderungen an die Verfügbarkeit. So sollen sie weder durch notwendige Wartungsarbeiten noch als Folge von Hardware- oder Softwareausfällen nicht erreichbar sein. Diese Server-Infrastruktur stellt also ein mögliches Einsatzgebiet der in dieser Arbeit betrachteten Hochverfügbarkeitslösungen dar und dient gleichzeitig als Testumgebung für deren Evaluierung.

Um das Ziel der Hochverfügbarkeit zu erreichen, gibt es wie eingangs der Arbeit erläutert unterschiedliche Ansätze. Im Grunde stimmen all diese Ansätze darin überein, dass sog. *Single Points of Failure* (kurz *SPOF*) eliminiert werden sollen. Dies erfolgt in aller Regel durch Bereitstellung von Backup-Systemen, die zum Einsatz kommen falls ein System ausfällt.

## 3.1 Architektur des Server-Pools

Der Server-Pool, der die Produktivdienste des Lehrstuhls bereitstellt, besteht im Wesentlichen aus einer Reihe von identischen Hardwarebausteinen, x86_64 Servern, von denen jeder mit vier CPU-Kernen ausgestattet ist. Zusammen mit einer ausreichenden Ausstattung an Hauptspeicher bilden sie die Basis für die komplett virtualisierte Service-Infrastruktur. Auf jedem der Server ist als Basis für die Virtualisierung ein mit Gentoo Linux erstelltes 64-Bit Xen Hostsystem (0-Domäne) in Betrieb. In dieser Umgebung werden die ebenfalls mit einem 64-Bit Gentoo Linux installierten Gastsysteme (U-Domänen), welche die eigentlichen Services bereitstellen, betrieben. Ein über FibreChannel an zwei der Server angebundenes, zentrales Storage-System hält sowohl System- als auch Nutzerdaten vor.

In den 0-Domänen laufen keine Serverdienste für Endbenutzer, um die Basisinstallation möglichst klein und übersichtlich zu halten; die privilegierten 0-Domänen dienen unter Xen auch als Managementumgebung für die virtualisierten Gastsysteme.

Das Host-Betriebssystem, d.h. die 0-Domäne, wird von den lokalen Festplatten (hardwarebasierter RAID-1) der jeweiligen Server gestartet, während die kompletten Root-Dateisysteme der Gastsysteme auf dem gemeinsam genutzten Speichersystem liegen. Einer der beiden an das zentrale Storage-System angebundenen Server fungiert ständig als NFS-Server für die Root-Dateisysteme der U-Domänen. Beim Booten einer U-Domäne mountet diese ihr Root-Dateisystem von diesem NFS-Server bevor das Gentoo-Gastsystem gestartet wird. Durch diese Art der zentralen Instal-

lation der Root-Dateisysteme können die U-Domänen auf beliebigen 0-Domänen, d.h. Servern, gestartet werden. Zusätzlich sind dadurch auch Features wie die Live-Migration von U-Domänen möglich.

Alle physikalischen Server sind mit jeweils vier GigaBit Ethernet Netzwerkkarten ausgestattet. Eine davon ist jeweils an das öffentliche Netz angebunden. Die restlichen drei *NIC*s wurden mit dem *Linux Channel Bonding* (siehe Abschnitt 2.3.3) zu einer ausfallsicheren Netzwerkschnittstelle zusammengefasst. Dieses private Netzwerk („Server-Netz“) ist vom öffentlichen Netz abgekapselt und somit nur für die 0-Domänen und die U-Domänen erreichbar. Da dieses Netz mit drei gebündelten GigaBit Leitungen sehr hohen Durchsatz ermöglicht, und aufgrund der Trennung vom öffentlichen Netzwerk, werden die Root-Dateisysteme der U-Domänen über dieses Netzwerk eingemountet.

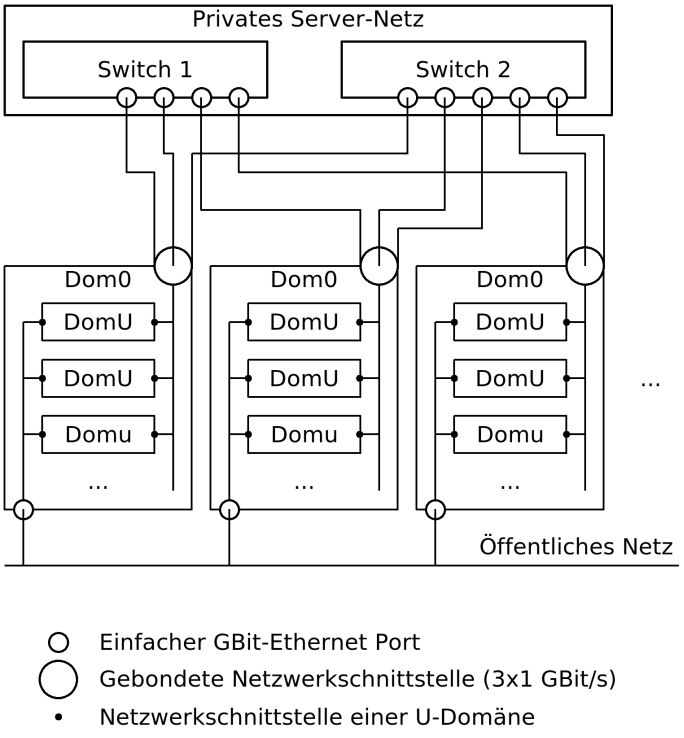


Abbildung 3.1: Redundanz im Server-Netz durch zwei Switches und Bonding

Den Gastsystemen werden von den 0-Domänen wiederum je drei virtuelle Netzwerkschnittstellen zur Verfügung gestellt. Über eine sind sie mit dem Server-Netz verbunden. Über die zweite Schnittstelle kann ihnen bei Bedarf eine globale IP-Adresse im öffentlichen Netz für den Zugriff aus dem Internet zugewiesen werden. Über die dritte Schnittstelle kann optional mit Hilfe von *Network Address Translation* (*NAT*) jedem Host Zugriff auf das Internet ermöglicht werden, ohne ihn direkt über eine globale IP-Adresse im öffentlichen Netz erreichbar zu machen.

Im Rahmen eines vorangegangenen Systementwicklungsprojektes wurde diese Plattform konzipiert und umgesetzt (siehe [16]). Sie beherbergt derzeit die gesamten Server-Dienste des Lehrstuhls.

## 3.2 Einsatzszenario und Single Points of Failure

Eine Hochverfügbarkeitslösung in diesem Szenario muss auf diese Netzwerkkonfiguration abgestimmt sein. Das gebündelte und somit ausfallsichere Server-Netz bietet sich beispielsweise an, um administrative Nachrichten wie Heartbeats zu verschicken. Des Weiteren kann eine HA-Lösung davon profitieren, dass jede U-Domäne auf allen Hosts lauffähig und somit unabhängig von der Serverhardware ist. Zusätzliche Features wie die Live-Migration von U-Domänen ermöglichen außerdem den unterbrechungsfreien Umzug von U-Domänen auf andere Hardware. Bei sich anbahnenden Hardwaredefekten ist es damit möglich, alle Gastsysteme einer 0-Domäne auf die restlichen 0-Domänen zu verteilen, ohne diese herunterfahren zu müssen. Zusätzlich sollten alle Knoten eines HA-Clusters auf verschiedene 0-Domänen verteilt werden, um die Server-Hardware als *Single Point of Failure* auszuschließen.

Das Mail-System des Lehrstuhls besteht aus insgesamt drei Subsystemen, bereitgestellt auf drei U-Domänen. Die erste (*Host 1*) ist zuständig für Mailversand und -zustellung. Der zweite (*Host 2*) scannt alle ein- und ausgehenden Mails nach Schadsoftware und der dritte (*Host 3*) verwaltet und archiviert die Mailinglisten.

*Host 1* mountet alle Benutzerdaten (d.h. die Spool-Dateien, in die die eingehenden Mails zugestellt werden, sowie die IMAP-Homes) vom Fileserver. *Host 3* mountet alle notwendigen Mailinglisten-Daten (die Archive, die Mailinglisten-Konfigurationen, ...) vom Fileserver. Aus diesem Aufbau des Mailsystems ergeben sich zahlreiche Abhängigkeiten, die bei der Konzeption einer HA-Lösung beachtet werden müssen.

In Abbildung 3.2 werden diese Zusammenhänge verdeutlicht.

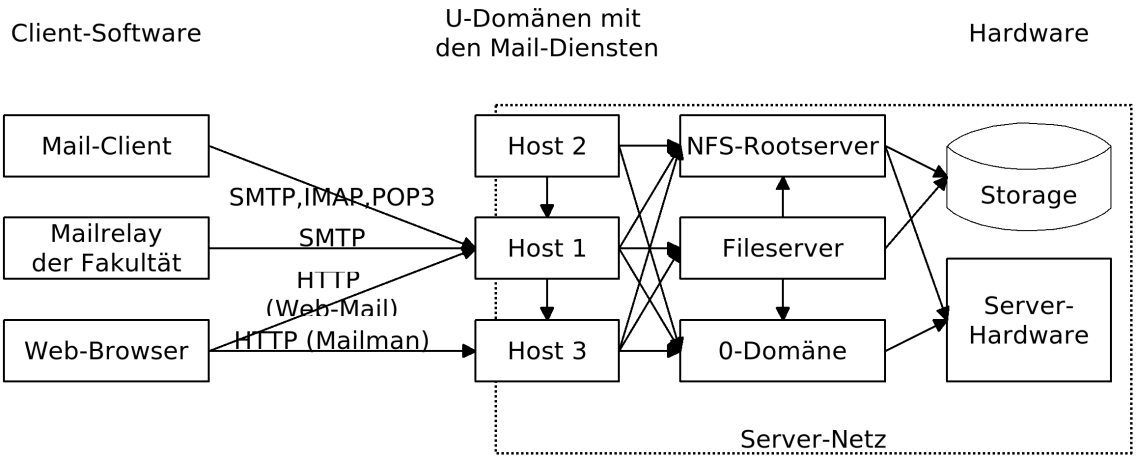


Abbildung 3.2: Das Mailsystem und seine Abhängigkeiten

Anhand dieses Beispiels ergeben sich aus der dargestellten Xen-Infrastruktur folgende *Single Points of Failure*, die zur Erhöhung der Verfügbarkeit eliminiert werden müssen:

**Dienste:** Die Dienste stellen die abstrakteste und aus Anwendersicht maßgebliche Stufe dieses Gesamtsystems dar. Die Interaktion des Anwenders mit dem Mail-

system erfolgt über die Dienste SMTP, IMAP, POP3 und HTTP (für Webmailer und Mailman-Frontend). Damit der Anwender in gewohnter Weise mit dem Mailsystem arbeiten kann, müssen all diese Dienste verfügbar sein und korrekt funktionieren.

**U-Domänen:** Damit alle Dienste funktionieren und erreichbar sind, müssen die virtuellen Xen-Maschinen, die diese Dienste bereit stellen (*Host 1* und *Host 3*) sowie der Contentfilter (*Host 2*) voll funktionsfähig sein.

**Fileserver:** Außerdem müssen der Fileserver (ebenfalls eine U-Domäne) und seine NFS-Freigaben erreichbar sein. Ist der Fileserver nicht mehr erreichbar, so ist kein Zugriff auf die Benutzerdaten mehr möglich.

**NFS-Rootserver:** Damit alle genannten virtuellen U-Domänen lauffähig sind, muss der NFS-Rootserver, der die Root-Dateisysteme der U-Domänen freigibt und in einer 0-Domäne läuft, ständig erreichbar sein.

**Server-Hardware:** Wenn die Hardware eines der Server ausfällt, sind gleichzeitig die U-Domänen betroffen, die auf der Hardware ausgeführt wurden. Diese können allerdings mit geringem Aufwand auf den verbleibenden Servern wieder gestartet werden.

**Speichersystem:** Im Falle eines Defektes des Speichersystems wären sowohl die Root-Dateisysteme aller Xen-Hosts als auch die Benutzerdaten nicht mehr verfügbar. Als Folge wären alle virtualisierten Dienste sowie sämtliche Dienste außerhalb der Xen-Infrastruktur, die Zugriff auf NFS-Freigaben benötigen, nicht mehr benutzbar.

Auch das Server-Netzwerk selbst ist ein wichtiger Bestandteil dieser Architektur, und ein Ausfall einzelner Komponenten des Netzes würde ebenfalls die Verfügbarkeit der Dienste negativ beeinflussen. Durch die eingesetzte Bonding-Technologie (siehe Abschnitt 2.3.3) ist das Server-Netz bereits ausfallsicher.

# 4 Bewertung und Auswahl

Im Hinblick auf einen möglichen Einsatz innerhalb der beschriebenen, existierenden Serverlandschaft sind aufgrund des eingesetzten Betriebssystems Gentoo Linux bereits viele Parameter festgelegt, die die Hochverfügbarkeitslösung erfüllen muss.

Die im Abschnitt 2.2 aufgeführten proprietären Lösungen sind für höchste Anforderungen im kommerziellen Einsatz konzipiert und scheiden schon allein aus Gründen der Lizenzkosten für eine Verwendung in einem universitären Serververbund aus. Ein weiteres Auswahlkriterium sollte die Verfügbarkeit der Softwarepakete in der Paket-Verwaltung der eingesetzten Linux-Distribution darstellen. Ein Ziel bei der Einrichtung der beschriebenen Serverlandschaft war es, eine einheitliche Infrastruktur für alle angebotenen Dienste bereitzustellen. Deshalb sollte dieses Ziel auch bei allen weiteren Projekten verfolgt werden. Ein wichtiges Kriterium hierfür ist die Möglichkeit einheitlicher und eventuell auch automatischer Software-Updates mit Hilfe des Gentoo-Paketsystems. Um auch künftig das Einspielen von Sicherheits- und Softwareupdates so einfach wie möglich zu gestalten, sollte deshalb Wert auf die Verfügbarkeit der eingesetzten Software im Paketsystem gelegt werden. Die meisten Lösungen aus Abschnitt 2.3 sind bereits unter Gentoo verfügbar. Die einzige Software, die hier weder über das Paketsystem, noch als Teil des Linux Kernels installiert werden kann, ist Crossroads (2.3.5).

Am vielversprechendsten erscheinen zunächst die Failover-Lösungen, mit denen sich den Produktivsystemen Standbysysteme zur Seite stellen lassen, da sich mit deren Hilfe viele SPOFs eliminieren lassen. In der in 3.1 beschriebenen Umgebung könnten mit der Hilfe von Heartbeat folgende SPOFs aus Abschnitt 3.2 beseitigt werden:

**Dienste:** Durch einen IP-Failover auf einen Host, der alle Dienste des Produktivsystems bereitstellt (für *Host 1* zum Beispiel SMTP, IMAP, POP3 und HTTP), kann eine Software wie Heartbeat alle diese Dienste mit minimaler Konfiguration hochverfügbar halten.

**U-Domänen:** Da Heartbeat auch in den 0-Domänen installiert werden kann, können auch die U-Domänen überwacht und im Fehlerfall migriert werden.

**Fileserver:** Da der Fileserver ebenfalls ein Dienst ist (NFS-Server), kann er ebenso wie die Dienste im ersten Punkt hochverfügbar gehalten werden. Da er zudem in einer separaten U-Domäne läuft, lässt sich zusätzlich die zweite Methode anwenden.

**NFS-Rootserver:** Für den NFS-Rootserver gilt dies nicht. Er ist lediglich ein Dienst. Da dieser in einer 0-Domäne, also in keinem migrierbaren Gastsystem läuft, lässt er sich ausschließlich durch die erste Failover-Methode gegen Ausfälle absichern.

Für eine Loadbalancing-Lösung benötigt man in aller Regel ebenfalls einen Failover-Mechanismus, um mit dem Loadbalancer keinen zusätzlichen SPOF zu schaffen. Da der Loadbalancer alle Anfragen an weitere Maschinen verteilt und dadurch die Anfragen parallel von verschiedenen Rechnern abgearbeitet werden, kann dieses Verfahren einen zusätzlichen Leistungsgewinn bedeuten.

Der Fileserver und der NFS-Rootserver können nicht mit Hilfe eines Loadbalancers redundant gemacht werden, da beide exklusiven Zugriff auf die Dateisysteme die sie freigeben (keine Cluster-Dateisysteme) benötigen.

Die Server-Hardware und das Speichersystem bleiben in der existierenden Konfiguration bei allen hier untersuchten Verfahren weitestgehend als SPOF bestehen, da Hardwareausfälle nicht durch diese Software abgefangen werden können. Das Bonding im Server-Netz macht allerdings drei der vier Netzwerkkarten jedes Servers ausfallsicher. Und die Möglichkeit zur einfachen Migration von U-Domänen ermöglicht schnelles Reagieren auf anbahnende Hardwaredefekte oder vereinfacht Wartungsarbeiten an den 0-Domänen und den Servern. Außerdem ist die Hardware selbst bereits redundant ausgelegt. Die Server und das Speichersystem besitzen jeweils redundante Netzteile, um bei Unterbrechungen der Stromversorgung eines Netzteils weiter arbeiten zu können. Das RAID 5 im Speichersystem gewährleistet zusätzlich eine gewisse Sicherheit gegen Ausfälle einzelner Festplatten. Weitergehende Sicherheit im Hinblick auf einen Ausfall des gesamten Speichersystems könnte durch ein zweites gleichartiges System geschaffen werden, und einen Replikationsmechanismus, wie z.B. eine Spiegelung der Daten beider Systeme auf SAN-Ebene.

Da alle für die Architektur des Gesamtsystems wichtigen Daten zentral gespeichert und über den Fileserver exportiert werden (siehe 3.1), wird keine Datenreplikation zwischen Rechnern benötigt. Replikations-Mechanismen wie NBD und DRBD wären nur dann notwendig, wenn Daten synchronisiert werden müssen, die nicht auf dem zentralen Storage liegen, beziehungsweise wenn das Storage als SPOF eliminiert werden soll, ohne (s.o.) einen SAN-basierten Replikationsmechanismus zu verwenden.

Für eine detailliertere Evaluierung im Rahmen einer Installation und testweisen Inbetriebnahme in dem beschriebenen Server-Pool wurden somit *Heartbeat* aus dem Linux-HA Projekt, *Linux Virtual Server* und *ClusterIP* ausgewählt. Deren Installation und Konfiguration ist Gegenstand des nachfolgenden Abschnitts.

# 5 Umsetzung

Wie bereits im ersten Kapitel angesprochen, ist für die Beseitigung aller *SPOFs* in einem System fast immer eine Failoverlösung notwendig, um die Verfügbarkeit einzelner Dienste oder auch kompletter Server zu erhöhen. Ansätze, die die Zuverlässigkeit durch Loadbalancing erhöhen wollen, führen in der Regel mit dem Loadbalancer einen neuen SPOF ein, oder benötigen zusätzlich einen Mechanismus, um Ausfälle von Clusterknoten festzustellen. Failover-Software ist also ein essentieller Bestandteil vieler hochverfügbarer Computersysteme.

## 5.1 Konventionen für die Beispiele

Für die Beispiele in den folgenden Abschnitten werden exemplarisch fiktive IP-Adressen verwendet. Insbesondere werden auch Beispiele auftauchen, in denen sowohl Mail- als auch HTTP-Dienste verwendet werden. Deshalb gelten für den Rest dieser Arbeit folgende Konventionen:

Die öffentliche IP-Adresse für die Mail-Dienste lautet `123.123.123.1`, die der HTTP-Dienste `123.123.123.2`. Im Abschnitt über Loadbalancing (Abschnitt 5.3) werden diese IP-Adressen konform zur IPVS-Terminologie auch als *virtual IP* oder *VIP* bezeichnet.

Als private IP-Adressen für die *Real Server*, die die Anfragen beim Loadbalancing entgegen nehmen, wurden für die Mailserver `192.168.0.101` und `192.168.0.102` verwendet. Die Webserver erhalten im privaten Netz entsprechend die IP-Adressen `192.168.0.201` und `192.168.0.202`.

Für die NAT-Gateways in Abschnitt 5.3 wurden die IP-Adressen `192.168.0.1` (für die Mailserver) und `192.168.0.2` (für die Webserver) verwendet.

Das Standard-Gateway für das Subnetz `123.123.123.0/24` sei unter der IP-Adresse `123.123.123.254` erreichbar.

## 5.2 Failover mit Heartbeat

Bereits 1999 veröffentlichte das Linux-HA Projekt die erste Version seiner Hochverfügbarkeitslösung mit dem Namen *Heartbeat*. Ende Juli 2005 erschien dann die zweite Version von Heartbeat. Die Entwickler versprachen sich davon noch konkurrenzfähiger zu kommerziellen Produkten zu sein. Eine wichtige Neuerung der zweiten Version ist, dass die Cluster nun nicht mehr auf nur zwei Knoten beschränkt sind,

sondern theoretisch Cluster beliebiger Größe aufgebaut werden können. In diesem Punkt ist Heartbeat bereits vielen kommerziellen Konkurrenzprodukten überlegen, da die meisten in der maximalen Clustergröße beschränkt sind. Eine weitere wichtige Neuerung von Heartbeat 2 ist die Möglichkeit Ressourcen zu überwachen. Im Gegensatz zu Heartbeat 1 können also nicht nur komplette Knoten auf ihre Funktionsfähigkeit überprüft werden, sondern auch einzelne Dienste die die Knoten anbieten.

### 5.2.1 Funktionsweise

Um den Ausfall eines Rechners zu erkennen verwendet Heartbeat sogenannte *Heartbeats* was auch die Namensgebung der Software erklärt. Bei den *Heartbeats* handelt es sich um UDP-Pakete, die jeder Knoten in regelmäßigen Zeitabständen emittiert. Alle anderen Knoten im HA-Cluster erwarten diese *Heartbeats* der anderen Knoten im Cluster. Bleiben sie aus, gilt der Knoten als ausgefallen und ein Failover seiner Dienste wird eingeleitet.

Um auch auf Ausfälle einzelner Dienste reagieren zu können, kann Heartbeat diese sogenannten Ressourcen mit Agenten überwachen. Jeder Dienst kann in Heartbeat als Ressource definiert werden und dann mit Hilfe von *Resource-Agents* gestartet, gestoppt und überwacht werden. Je nach Konfiguration kann Heartbeat dann ausgefallene Ressourcen neu starten, oder auf andere Clusterknoten migrieren.

### 5.2.2 Architektur von Heartbeat

Die Heartbeat-Software besteht aus mehreren Komponenten. Die wichtigsten sind:

**Heartbeat:** Diese Kernkomponente stellt diverse Kommunikationsmechanismen bereit. Zum einen die Heartbeats, mit denen Knoten die Ausfälle anderer Knoten erkennen können. Zusätzlich werden aber auch administrative Nachrichten über diese Komponente ausgetauscht. Beispielsweise wird die Clusterkonfiguration über die Heartbeatkomponente an alle anderen Clusterknoten verteilt.

**CRM:** Der *Cluster Resource Manager* verwaltet die Ressourcen. Er entscheidet beispielsweise welche Ressource auf welchem Knoten gestartet bzw. gestoppt wird. Der CRM ist eine zentrale Komponente in der Heartbeat Architektur. Er interagiert direkt mit den meisten anderen Komponenten.

**CIB:** Die *Cluster Information Base* repräsentiert den Zustand des Clusters. Die CIB wird auf alle Knoten im Cluster repliziert. Sie enthält Informationen darüber, aus welchen Knoten der Cluster besteht und welche Ressourcen auf welchem Knoten laufen. Die CIB wird im XML-Format gespeichert und kann auf jedem Knoten direkt angezeigt werden.

**LRM:** Der *Local Resource Manager* startet, stoppt und überwacht die lokalen Ressourcen eines Knotens. Er wird direkt vom CRM gesteuert.

**Resource Agents:** Die *Resource-Agents* sind die Schnittstelle zwischen dem LRM und den Diensten. Hierbei handelt es sich um Skripten, mit denen ähnlich

wie mit den Unix-typischen Init-Skripten Dienste gestartet, gestoppt bzw. deren Status abgefragt werden kann. Es können neben OCF und Heartbeat v1-konformen Skripten auch die LSB konformen Init-Skripten des Betriebssystems als *Resource-Agents* verwendet werden.

**STONITH Dämon:** Ermöglicht es den anderen Knoten im Cluster Dienste oder den ganzen Knoten zu resetten.

**Logd:** Ein eigener Logging-Dämon, der entweder über den Syslog-Dämon, direkt in Dateien oder über beide Methoden loggen kann. Er wurde eingeführt, um nicht mehr von blockierenden Syslog-Dämonen abhängig zu sein.

### Cluster Information Base (CIB)

Da fast die gesamte Clusterkonfiguration in Heartbeat 2 in der CIB gespeichert wird, wird diese im Folgenden näher erklärt. Die komplette CIB wird im XML-Format (unter `/var/lib/heartbeat/crm/cib.xml`) gespeichert und verarbeitet. Das Hinzufügen neuer Ressourcen, wie auch die Modifikation von Teilen der CIB erfolgt ebenfalls über Anfragen im XML-Format. Eine vollständige CIB enthält folgende Abschnitte:

`crm_config`**:** Speichert die Konfiguration des Clusters.

`nodes`**:** Eine Liste aller Clusterknoten.

`resources`**:** Die Ressourcen, die auf die Clusterknoten verteilt werden.

`constraints`**:** Abhängigkeiten der Ressourcen untereinander.

Eine leere CIB sieht wie folgt aus:

```
<cib>
   <configuration>
     <crm_config/>
     <nodes/>
     <resources/>
     <constraints/>
   </configuration>
   <status/>
</cib>
```

### Ressourcen

Als Ressourcen werden bei Heartbeat die Dienste bezeichnet, die durch Heartbeat überwacht werden, und die dessen Kontrolle unterliegen. Definiert werden diese Ressourcen im `<resources/>` Abschnitt der CIB. Überwacht, gestartet und gestoppt werden die Ressourcen durch den LRM mit Hilfe von Skripten. Hierfür können entweder die Init-Skripten des Systems (unter `/etc/init.d`), oder Skripten, die den OCF-Standard (*Open Cluster Framework*) erfüllen, verwendet werden. Voraussetzung dafür, dass sich die System-Skripten verwenden lassen, ist dass sich diese an

den LSB Standard für Rückgabewerte der `start()`, `stop()` und `status()` Funktionen halten. Einige *OCF-Resource-Agents* sind bereits Teil des Heartbeat Pakets. Diese befinden sich in dem Verzeichnis `/usr/lib/ocf/resource.d/heartbeat`. Es können aber auch eigene OCF-konforme *Resource-Agents* verwendet werden. Diese müssen dann in einem Unterverzeichnis von `/usr/lib/ocf/resource.d` abgelegt werden.

Die XML-Definition einer Ressource erfolgt nach folgendem Schema:

```
<resources>
 <primitive class="ocf" id="IP-Adresse" provider="myResources"
         type="IPaddr">
  <instance_attributes>
   <attributes>
    <nvpair id="ip" name="ip" value="123.123.123.1"/>
    <nvpair id="mask" name="netmask" value="24"/>
    <nvpair id="nic" name="nic" value="eth0"/>
    <nvpair id="gw" name="gateway" value="123.123.123.254"/>
   </attributes>
  </instance_attributes>
 </primitive>
</resources>
```

Hier wird eine OCF-Ressource vom Typ `IPaddr` definiert. Das `provider`-Attribut gibt den Namen des Unterverzeichnisses in `/usr/lib/ocf/resource.d` an, in dem nach dem *OCF-Resource-Agent* gesucht wird. Das Attribut `type` gibt den Namen des Skriptes an. Die Zeile

```
<primitive class="ocf" id="IP-Adresse" provider="myResources"
        type="IPaddr">
```

weist Heartbeat an, für die Ressource mit der ID `IP-Adresse` den *OCF-Resource-Agent* `/usr/lib/ocf/resource.d/myResources/IPaddr` zu verwenden. Über den `<instance_attributes/>` Block können dem *Resource-Agent* Parameter übergeben werden. Dies geschieht durch die *nvpair*-Elemente. Der Wert des XML-Tags `<nvpair id="ip" name="ip" value="123.123.123.1"/>`, also die IP-Adresse `123.123.123.1`, ist im Skript des *Resource-Agents* unter dem Variablennamen `$OCF_RESKEY_ip` abrufbar.

Das konkrete Beispiel definiert eine IP-Adresse als Heartbeat-Ressource. Der Knoten auf dem die Ressource aktiv ist, ist dann unter der IP-Adresse `123.123.123.1` erreichbar. Migriert man die Ressource von einem Knoten auf einen anderen, wird sie zunächst auf dem ersten Knoten gestoppt.

### Operationen

Über Operationen können verschiedene Eigenschaften der `start`, `stop` und `monitor` Aktionen festgelegt werden. So können Vorbedingungen, Timeouts und das Ver-

halten des Clusters beim Scheitern der Aktion konfiguriert werden. Für `monitor`-Operationen kann zusätzlich das Intervall eingestellt werden, in dem die Ressource überprüft werden soll.

Für jede Operation einer Ressource lässt sich individuell festlegen, was bei Fehlern geschieht. Die Ressource kann dadurch neu gestartet, gestoppt oder der ganze Knoten auf dem die Ressource lief isoliert, und alle Ressourcen von ihm weg migriert werden.

#### Ressourcengruppen

Mehrere Ressourcen können zu Gruppen zusammengefasst werden. Fällt eine Ressource einer Gruppe aus, wird entweder die komplette Gruppe (d.h. all ihre Ressourcen) auf einen anderen Knoten migriert oder alle Ressourcen, die nach (d.h. unterhalb) der ausgefallenen Ressource definiert wurden, werden gestoppt. Danach wird die ausgefallene Ressource wieder gestartet, und anschließend auch die darauf folgenden Ressourcen.

Mit Hilfe der Ressourcengruppen können beispielsweise IP-Adresse, NFS-Mounts, SMTP-Server und IMAP-Server zusammengefasst werden. Sobald eine dieser Ressourcen ausfällt wird die Gruppe als Ganzes auf ein Standbysystem umgezogen.

Definiert werden Gruppen als mehrere Ressourcen, die von einem `<group>`-Tag eingeschlossen werden:

```
<resources>
 <group id="mailserver">
  <primitive id="resource-1"/>
  <primitive id="resource-2"/>
  <primitive id="resource-3"/>
 </group>
</resources>
```

Zu beachten ist die Reihenfolge, in der die Ressourcen definiert werden. Im obigen Beispiel ist die `resource-1` eine Voraussetzung für `resource-2`. Diese ist wiederum eine Vorbedingung für die Ressource `resource-3`. Das bedeutet, dass die Ressourcen in der Reihenfolge ihrer Definitionen gestartet werden. Wenn eine Ressource nicht startet, so werden auch die darauf folgenden Ressourcen nicht gestartet. Fällt eine Ressource aus, werden alle darauffolgenden Ressourcen solange deaktiviert, bis die fehlerhafte Ressource wieder erfolgreich gestartet wurde.

### 5.2.3 Installation und Konfiguration

#### Installation

Da Heartbeat im Gentoo Portage-Tree noch als `~amd64`, also als unstable für die x86_64 Architektur markiert ist, ist es nötig Heartbeat vor der Installation als `stable`

zu markieren. Dies erfolgt mit dem Kommando:

```
# echo "=sys-cluster/heartbeat-2.0.8 ~amd64" >> \
        /etc/portage/package.keywords
```

Auf der Linux-HA Homepage ([18]) gilt Heartbeat ausdrücklich als stabil auf 64-Bit Architekturen. Die Demaskierung sollte deshalb ohne Bedenken möglich sein.

Nachdem die Software übersetzt und installiert wurde, müssen noch mehrere Konfigurationsdateien editiert werden.

#### Konfiguration

Die Grundkonfiguration von Heartbeat erfolgt in drei Dateien:

`/etc/ha.d/ha.cf`: Hier werden vor Allem die Netzwerkparameter für die Heartbeat-Verbindung zwischen den Cluster-Knoten konfiguriert.

`/etc/ha.d/ha_logd.cf`: Heartbeat besitzt einen eigenen Logdämon, um es vom Syslog-Dämon zu entkoppeln. Dieser wird über diese Datei konfiguriert.

`/etc/ha.d/authkeys`: Die Authentifizierung der Cluster-Knoten untereinander wird über diese Datei konfiguriert.

Im Anhang finden sich Beispiele dieser Dateien. Auch die einzelnen Optionen in den Konfigurationsdateien werden dort erläutert (siehe Anhang B.1). Diese Beispiele sind für einen Cluster, der aus zwei Knoten besteht, ausgelegt. Auf jedem Knoten muss eine identische Kopie der Dateien liegen.

#### Starten des Heartbeat-Dämons

Nach erfolgter Konfiguration kann der Cluster in Betrieb genommen werden. Hierfür muss Heartbeat auf den Cluster-Knoten über das mitgelieferte Init-Skript gestartet werden:

```
# /etc/init.d/heartbeat start
```

Über die distributionsspezifischen Konfigurationswerkzeuge (in Gentoo `rc-update`) lässt sich Heartbeat dann im Standard-Runlevel starten.

### 5.2.4 Verwaltung und Überwachung des Clusters

Nachdem der Cluster erfolgreich gestartet wurde, kann dieser über verschiedene Dienstprogramme konfiguriert werden.

### Verwaltung der CIB

Das Programm `cibadmin` ist sehr vielseitig. Mit seiner Hilfe lässt sich die CIB anzeigen, löschen, erweitern, ändern sowie eine Synchronisation der CIBs aller Knoten anstoßen.

Die Parameter, die hierfür benötigt werden, lauten

`cibadmin --cib_erase`: Die Komplette CIB leeren.

`cibadmin --cib_query`: Die aktuelle CIB vollständig ausgeben.

`cibadmin -obj_type resources --cib_create --xml-file ip.xml`: Erstellt aus der XML-Definition in der Datei `ip.xml` eine neue Clusterressource.
Bedeutung der einzelnen Parameter:

- `--obj_type`: Der Objekttyp der bearbeitet werden soll (`nodes`, `resources`, `constraints`, `crm_config` oder `status`).
- `--cib_create`: Einen neuen Abschnitt in der CIB erstellen.
- `--xml-file`: Datei in der die XML-Daten des zu erstellenden Objektes stehen.

### Ressourcen verwalten

Für die Verwaltung der Clusterressourcen existiert ein separates Tool. Damit lassen sich Ressourcen auflisten, löschen, lokalisieren, migrieren, sowie deren Parameter verändern.

`crm_resource --list`: Zeigt eine Liste aller Clusterressourcen an.

`crm_resource --delete --resource-type primitive --resource ip`: Löscht die Ressource mit dem Namen `ip` aus der CIB.

`crm_resource --migrate postfix`: Migriert die Ressource `postfix` weg vom derzeitigen Knoten. Die Ressource wird solange nicht mehr auf diesem Knoten gestartet, bis die Migration mit `crm_resource --un-migrate postfix` rückgängig gemacht wird.

### CIB-Code verifizieren

Zum Verifizieren und Debuggen von XML-Dateien kann das Programm `crm_verify` hilfreich sein.

`crm_verify --verbose --xml-file ip.xml`: Überprüft, ob die Datei `ip.xml` der *Document Type Definition (DTD)* (siehe [1]) entspricht.

`crm_verify --verbose --live-check`: Verbindet sich mit dem Cluster und verifiziert die aktuelle CIB gegen die DTD.

Sollte es beim Einspielen von XML-Daten in den CRM Probleme geben, lassen sich diese mit `crm_verify` finden.

#### Überwachen des Clusters

Das Tool `crm_mon` wird dazu verwendet, den aktuellen Cluster-Status abzufragen. Wird `crm_mon` ohne Parameter aufgerufen, aktualisiert sich die Statusanzeige alle 15 Sekunden. Über den Parameter `--interval` lässt sich das Intervall einstellen, nach dem sich die Ansicht aktualisieren soll.

Außerdem lässt sich das Tool mit der Option `--daemonize` als Dämon starten und die Ausgabe mit `--as-html` als HTML formatieren. Mit

```
# crm_mon --daemonize --as-html /export/html/cluster.html
```

kann also ein Dämon gestartet werden, der regelmäßig den Clusterstatus in eine HTML-Datei schreibt. Diese könnte dann wiederum über einen Webserver abgerufen werden.

### 5.2.5 Resource-Agents

#### IP-Failover

Im Softwarepaket von Heartbeat sind mehrere *Resource-Agents* enthalten. Neben Agenten für Apache, MySQL und Xen finden sich unter anderem auch Agenten, mit deren Hilfe IP-Adressen als Cluster-Ressourcen verwendet werden können.

Einer davon heißt `IPAddr`. Hiermit können zu bereits existierenden Netzwerkschnittstellen im System weitere virtuelle Netzwerkschnittstellen gestartet werden, denen dann eine IP-Adresse zugewiesen wird.

Für die Anforderungen in der beschriebenen Umgebung (siehe Abschnitt 3.1) eignet sich dieses Skript nur bedingt, da es auf einer bereits existierenden Netzwerkschnittstelle im System aufsetzt. Das würde zur Folge haben, dass jeder Knoten im Cluster bereits eine Schnittstelle mit einer öffentlichen IP-Adresse aktiviert haben muss (beispielsweise `eth0` mit `123.123.123.50`), bevor sich eine virtuelle Schnittstelle mit der gewünschten IP-Adresse (z.B. `eth0:0` mit `123.123.123.1`) starten lässt.

Da in der vorliegenden Konfiguration die U-Domains auch über ein privates Netzwerk erreichbar sind, ist es nicht notwendig jedem Knoten im Cluster eine öffentliche IP-Adresse zuzuweisen. Aus diesem Grund wurde der `IPaddr` OCF-Resource-Agent so modifiziert, dass er beim Failover zunächst das `eth0`-Interface ohne IP-Adresse startet, auf dem dann ein virtuelles Interface (also z.B. `eth0:0`) mit der betroffenen IP-Adresse gestartet wird. Dadurch werden im Cluster nur IP-Adressen verwendet, hinter denen sich tatsächliche Dienste verbergen. Es werden also keine IP-Adressen dafür verbraucht, dass die eth0-Schnittstelle ständig auf allen Knoten aktiv ist. Eine Anpassung des Init-Skripts `/etc/init.d/net.eth0`, um das eth0-Interface ohne IP zu starten, ist damit nicht notwendig.

Ein IP-Failover erfordert auf dem Zielrechner das Ausführen folgender Kommandos:

```
# ifconfig eth0 up
```

```
# ifconfig eth0:0 123.123.123.1 broadcast 123.123.123.255 \
        netmask 255.255.255.0 up
# route add default gw 123.123.123.254 eth0
```

Der zweite Befehl wird bereits durch den vorhandenen `IPaddr` *Resource-Agent* in der folgenden Zeile ausgeführt:

```
CMD=‘‘$IFCONFIG $iface $ipaddr netmask \
        $netmask broadcast $broadcast’’;;
```

Ersetzt man diese Zeile durch

```
CMD=‘‘$IFCONFIG $iface_base up && $IFCONFIG $iface $ipaddr netmask \
        $netmask broadcast $broadcast’’;;
```

so wird zunächst das `eth0`-Interface gestartet, bevor das virtuelle Interface `eth0:0` gestartet wird.

Zum Setzen der Route wird außerdem die Gateway-Adresse benötigt. Deshalb wurde dem `IPaddr`-Agent ein zusätzlicher Parameter hinzugefügt. Über diesen Parameter (`OCF_RESKEY_gateway`) kann dem Skript das Gateway übergeben werden. Damit die Default-Route tatsächlich neu gesetzt wird, muss das Skript des `IPaddr` *Resource-Agents* so modifiert werden, dass an entsprechender Stelle das Kommando

```
CMD="$ROUTE add default gw $gateway $iface_base";;
```

ausgeführt wird.

Alle Änderungen am IPaddr-Skript finden sich im Anhang (siehe B.2).

Eine IP-Adresse lässt sich mit dem modifizierten *Resource-Agent* nach folgendem Schema als Ressource (z.B. als XML-Datei in `~/ip.xml`) definieren:

```
<resources>
 <primitive class="ocf" id="ip_resource" provider="myResources" \
        type="IPaddr" is_managed="true">
  <instance_attributes id="ip_attributes">
   <attributes>
    <nvpair id="ip_ip" name="ip" value="123.123.123.1"/>
    <nvpair id="ip_mask" name="netmask" value="24"/>
    <nvpair id="ip_nic" name="nic" value="eth0"/>
    <nvpair id="ip_gw" name="gateway" value="123.123.123.254"/>
   </attributes>
  </instance_attributes>
 </primitive>
</resources>
```

Mit

```
# cibadmin --obj_type resources --cib_create --xml-file ~/ip.xml
```

kann die Ressource in der CIB erstellt werden.

### Failover einzelner Dienste

Laut Dokumentation müssen die Init-Skripten die LSB-Spezifikation erfüllen (siehe [12]). Für die Skripten von Gentoo ist dies allerdings nicht der Fall, da die Statusabfrage für gestoppte Dienste `1` liefert, aber um OCF-kompatibel zu sein `3` liefern müsste (siehe [12]).

Mit Hilfe eines OCF-konformen Wrapper Skripts (siehe Anhang B.3) kann dieses Problem umgangen werden. Übergibt man diesem *Resource-Agent* (`GentooService`) als Parameter den Namen eines Gentoo-Init-Skripts, so startet und stoppt dieser *Resource-Agent* je nach Bedarf diesen Dienst. Ohne zusätzlichen Aufwand lassen sich hiermit Dienste wie Postfix, UW-IMAP und Apache von Heartbeat steuern.

Der entsprechende CIB-Eintrag einer `GentooService`-Ressource könnte dann folgendermaßen aussehen:

```
 <primitive class="ocf" id="postfix" provider="myResources" \
        type="GentooService" is_managed="true">
 <operations>
  <op id="postfix_monitor" name="monitor" interval="5s"/>
 </operations>
 <instance_attributes id="postfix-attributes">
  <attributes>
   <nvpair id="postfix_service" name="service" value="postfix"/>
  </attributes>
 </instance_attributes>
</primitive>
```

### Mounten von Dateisystemen

Um Dateisysteme zu mounten existiert ebenfalls ein vorgefertigter *OCF-Resource-Agent*. Da in der Praxis gelegentlich Probleme beim Unmounten von NFS-Mounts auftreten, wurde auch dieser Agent modifiziert. Gelingt das Unmounten des NFS-Shares nicht, so wird noch ein Versuch mit der `umount`-Option `-l` unternommen. Dadurch wird ein „lazy unmount“ durchgeführt. Es können dann keine neuen Prozesse mehr Dateideskriptoren auf dem Dateisystem, das ausgehängt werden soll, öffnen. Wenn dann alle Dateideskriptoren geschlossen sind, wird das Dateisystem ausgehängt.

Die Änderungen am `Filesystem` Resource-Agent finden sich im Detail im Anhang B.4.

#### Beispiel für die CIB eines Mail-Servers

Mit Hilfe der modifizierten bzw. neu erstellten *Resource-Agents* lassen sich bereits alle drei Mailhosts aus Abschnitt 3.2 gegen Ausfälle absichern. Die User-Homes (`/home`) und das Spool-Verzeichnis (`/var/spool`) können vom NFS-Server mit Hilfe des `Filesystem` *Resource-Agents* eingebunden werden. Das Starten und Stoppen des Postfix- und UW-IMAP-Dämons übernimmt der `GentooService` Resource-Agent. Die *Floating IPs*, unter denen der Mailserver von den Clients aus (`123.123.123.1`) und innerhalb des privaten Netzes (`192.168.0.1`) erreichbar ist, werden bei Bedarf vom `IPaddr` *Resource-Agent* migriert.

Damit sich immer alle Ressourcen auf dem gleichen Knoten befinden, werden sie zu einer Gruppe zusammengefasst.

Die XML-Definitionen dieser Gruppe findet sich ebenfalls im Anhang B.5.

Die Dienste aus der Übersicht im Abschnitt 3.2 lassen sich dank dieser Lösung als *SPOF* eliminieren.

#### Beispiel für die CIB eines NFS-Servers

Für einen NFS-Server muss Heartbeat zunächst das Speichersystem mounten. Dies ist wiederum mit dem `Filesystem` Resource-Agent möglich. Als einziger Dämon muss vom `GentooService` Resource-Agent lediglich der NFS-Dienst selbst gestartet werden. Danach können die IP-Adressen für das private Netzwerk (`192.168.0.3`) sowie die öffentliche IP-Adresse (`123.123.123.3`) aktiviert werden.

Auch diese Beispielkonfiguration findet sich im Anhang (siehe Anhang B.6).

Durch diese Konfiguration können die *SPOFs* Fileserver und NFS-Rootserver aus Abschnitt 3.2 eliminiert werden.

Verteilt man zusätzlich die Aktiv- und Passiv-Syteme aus diesen beiden Beispielen auf verschiedene Server, so stellt auch die Server-Hardware keinen *SPOF* mehr dar. Da zusätzlich die Heartbeat-Instanzen in unterschiedlichen U-Domänen ausgeführt werden, sind auch die U-Domänen kein *SPOF* mehr. Der einzige *SPOF* aus 3.2, der durch ein Failover-System nicht abgedeckt wird, ist das Speichersystem.

## 5.3 Loadbalancing mit Linux Virtual Server

Das Projekt Linux Virtual Server (kurz LVS) besteht aus zwei Software-Komponenten. Der IPVS-Netfilter-Code ist bereits in den Linux-Kernel integriert. IPVS übernimmt das eigentliche Loadbalancing der eintreffenden Pakete. Die zweite Komponente stellt das Verwaltungstool `ipvsadm` dar, mit dem das Loadbalancing aufgesetzt und konfiguriert wird.

Ein einfacher LVS-Cluster ist allerdings nicht hochverfügbar. Ausfälle beim Loadbalancer (bei LVS *Director*) selbst hätten zur Folge, dass keiner der lastverteilten

Dienste mehr erreichbar wäre. Bei Ausfällen an den Maschinen, die die Dienste selbst bereit stellen (in LVS als *Real Server* bezeichnet), bleiben die Anfragen unbeantwortet, die vom *Director* an diesen *Real Server* weitergeleitet werden. Erfolgt beispielsweise das Loadbalancing in einem Cluster mit drei *Real Servern* nach dem Round Robin Verfahren, würde beim Ausfall eines *Real Servers* jede dritte Anfrage an den Cluster unbeantwortet bleiben.

Um Loadbalancing auch als Hochverfügbarkeitslösung einsetzen zu können, wird deshalb ein zweiter Loadbalancer sowie ein Mechanismus, der die oben genannten Ausfälle erkennt, benötigt. Einen solchen Mechanismus stellt die Software Keepalived bereit. Mit Hilfe dieser Software können die Loadbalancer zum Einen Ausfälle anderer *Directors* erkennen und zum Anderen können mit deren Hilfe auch die einzelnen Dienste der *Real Server* überwacht werden.

Ein solches System mit Keepalived ist in Abbildung 5.1 skizziert.

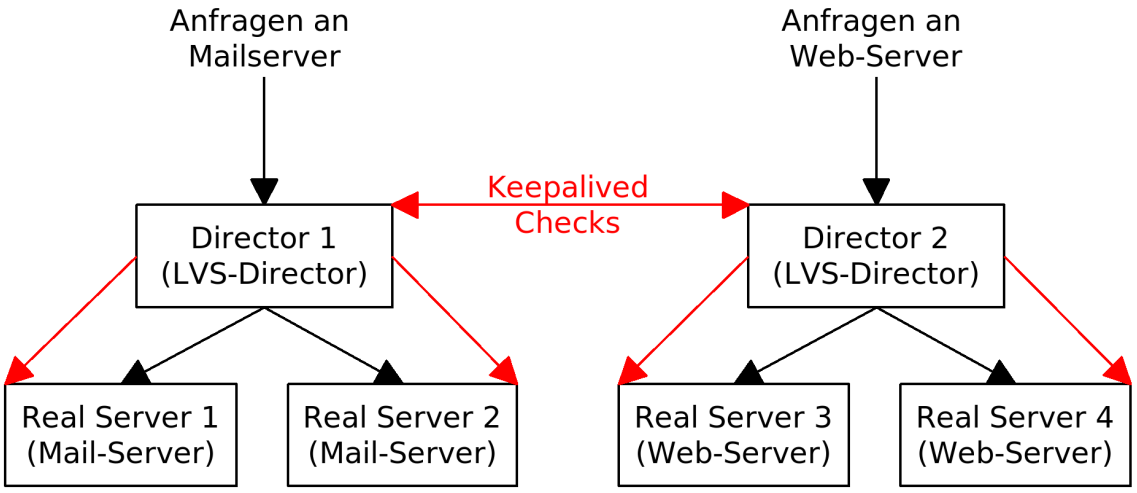


Abbildung 5.1: Ein hochverfügbarer LVS-Cluster mit Keepalived

Einer der beiden Loadbalancer fungiert als *Director* für den hochverfügbaren Dienst (z.B. Mail-Service). Der zweite Loadbalancer, der in diesem Fall nur zum Einsatz käme, wenn der erste ausfällt, kann verwendet werden um einen weiteren Dienst auf gleiche Weise hochverfügbar zu halten. Im Normalfall reagiert der *Director 1* auf alle Anfragen, die an die öffentliche IP-Adresse des Mailservers gerichtet sind und *Director 2* auf alle an die IP-Adresse des Webservers. Diese Anfragen leiten sie dann entsprechend des gewählten Loadbalancing-Algorithmus an jeweils mindestens zwei *Real Server* weiter, die die Anfrage dann bearbeiten und anschließend beantworten. Für die Weiterleitung der Pakete vom *Director* zu den *Real Servern* stellt LVS drei verschiedene Mechanismen zur Verfügung:

**LVS/NAT:** Der Client schickt die Anfragen an den Loadbalancer. Dieser wählt den *Real Server* nach dem vorgegebenen Algorithmus aus. Anschließend übersetzt er die IP-Adresse des Empfängers der Anfrage in die private IP-Adresse des ausgewählten *Real Servers*. Dieser schickt seine Antwort ebenfalls über den *Director* zurück an den Client. Die *Real Server* sind bei diesem Verfahren

komplett isoliert von den Clients, und nur im privaten Netzwerk erreichbar.

**LVS/DR:** Ähnlich wie bei LVS/NAT empfängt auch beim *Direct Routing* der *Director* zunächst alle Anfragen von den Clients. Anders als bei LVS/NAT werden die Adressen der IP-Pakete jedoch nicht umgeschrieben, sondern direkt an die MAC-Adressen der *Real Server* geroutet. Dort muss eine Dummy-Schnittstelle mit der *VIP* konfiguriert sein, damit die Pakete korrekt angenommen und verarbeitet werden. Die Antworten schicken die *Real Server* direkt an den Client.

**LVS/TUN:** Bei dieser Technik werden die beim *Director* eintreffenden Anfragen durch *IP-Tunneling* an die *Real Server* weiter gereicht. Dazu werden vom *Director* die eintreffenden IP-Pakete in ein neues IP-Paket eingebettet (ein sogenanntes IPIP-Paket), das dann an den entsprechenden *Real Server* adressiert ist. Auf den *Real Servern* muss dann aus dem IPIP-Paket das ursprüngliche Paket extrahiert werden. Hierfür muss der Kernel des *Real Servers IP-Tunneling* unterstützen. Außerdem muss ein *IP-Tunneling*-Interface (z.B. `tunl0`) mit der *VIP* konfiguriert sein. Wie bei LVS/DR werden die Antworten direkt an den Client geschickt.

Für LVS/TUN und LVS/DR benötigen die *Real Server* zusätzlich zur privaten Netzwerkschnittstelle auch eine öffentliche Schnittstelle, über die sie die Antworten an die Clients senden können. Dabei muß auf den *Real Servern* für die *VIP* eine Tunnel- bzw. Loopback-Schnittstelle eingerichtet werden. Bei LVS/NAT ist dies nicht notwendig, da alle IP-Pakete über den *LVS-Director* verschickt werden. Da die *Real Server* hier keine Netzwerkschnittstellen in ein öffentliches Netz besitzen, und nur noch über fest definierte Ports der *VIP* erreichbar sind, sind sie vom Internet, und somit von potentiellen Angriffen, entkoppelt.

Die möglichen Scheduling-Algorithmen, nach denen der *Director* die Anfragen an die *Real Server* verteilt, sind für alle drei LVS-Techniken gleich. Die Anfragen können z.B. anhand einer Round Robin-, Least-Connection-, Source-Hashing- oder Destination-Hashing-Strategie verteilt werden. Bei allen drei Lösungen speichert der *Director* die Verbindungen in einer Hash-Tabelle ab, um alle Pakete einer Verbindung an den gleichen *Real Server* schicken zu können. Nach einem Timeout werden die Einträge wieder aus der Hash-Tabelle entfernt (sog. persistente Verbindung).

### 5.3.1 Notwendige Anpassungen der Firewalls

Bei LVS/DR nimmt der *Director* alle Anfragen an die *VIP* des Clusters entgegen und routet sie an die MAC-Adresse des ausgewählten *Real Servers* weiter. Das bedeutet, dass alle Firewalls, die das Paket vom *Director* bis zum *Real Server* durchläuft (also Firewall des *Directors*, Firewall der 0-Domäne in der der *Director* läuft, Firewall der 0-Domäne, in der der *Real Server* läuft, und Firewall der *Real Servers* selbst), so konfiguriert werden müssen, dass beliebige Quell-Adressen der IP-Pakete erlaubt sind.

Das gleiche gilt für LVS/NAT. Da die *Real Server* hier jedoch ihre Antworten ebenfalls über das private Subnetz versenden, müssen zusätzlich Pakete mit beliebiger

Ziel-Adresse freigeschaltet werden.

In der beschriebenen Xen-Umgebung des Lehrstuhls müssen für LVS/NAT und LVS/DR die Firewalls aller 0-Domänen (siehe Anhang B.7) sowie, falls aktiv, die der *Real Server* umkonfiguriert werden. Da es sich bei den *Directors* um eine Gentoo-Basis-Installation ohne aktiver Firewall handelt sind hier keine Anpassungen erforderlich.

### 5.3.2 Konfiguration für LVS/NAT

Zur Konfiguration von LVS/NAT sind mehrere Schritte notwendig:

**NATting aktivieren:** Am *Director* muss iptables für NATting konfiguriert werden.

**Konfiguration von IPVS:** Am *Director* müssen die virtuellen Dienste, für die die Lastverteilung aktiviert werden soll, und die *Real Server*, an die die Anfragen verteilt werden, konfiguriert werden.

**Konfiguration der Real Server:** An den *Real Servern* muss das Standardgateway neu gesetzt und, falls vorhanden, die aktiven öffentlichen Netzwerkschnittstellen deaktiviert werden.

#### Aktivierung von NATting auf dem Director

Damit NATting funktioniert, muss zunächst auf dem Director `ip_forward` aktiviert werden, um IP-Pakete auf andere Netzwerkschnittstellen umleiten zu können. Die dafür notwendigen Kommandos lauten:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
for f in /proc/sys/net/ipv4/conf/*/rp_filter; do echo 0 > $f; done
```

Um das *NATting* selbst zu aktivieren sind weitere Kommandos notwendig:

```
iptables -A FORWARD -i eth1 -s 192.168.0.0/24 -j ACCEPT
iptables -t nat -A POSTROUTING -o eth0 -s 192.168.0.0/24 -j SNAT \
        --to-source 123.123.123.1
```

Das erste Kommando bewirkt, dass IP-Pakete, die an der Netzwerkschnittstelle eth1 eintreffen und aus dem Subnetz `192.168.0.0/24` stammen, weiter geroutet werden dürfen.

Das zweite Kommando bewirkt, dass bei IP-Paketen, die über die Netzwerkschnittstelle `eth0` verschickt werden und aus dem Subnetz `192.168.0.0/24` stammen, die Absenderadresse in `123.123.123.1` (also die *VIP*) umgeschrieben wird.

Das bedeutet, dass durch diese iptables-Regeln nur ausgehende IP-Pakete umgeschrieben werden (d.h. IP-Pakete die die Real Server an den Client schicken). Das NATting für die eintreffenden Pakete übernimmt IPVS selbst, da bei eintreffenden Paketen die IP-Adresse des Empfängers erst feststeht nachdem IPVS entschieden hat an welchen *Real Server* die Anfrage weitergeleitet wird.

In Anhang B.8 findet sich ein Beispiel für ein Firewall-Skript, das diese Schritte automatisch ausführt.

### Einrichten von IPVS auf dem Director

Die Konfiguration von IPVS erfolgt mit Hilfe der Anwendung `ipvsadm`. Um einen neuen virtuellen Dienst in IPVS anzulegen stellt `ipvsadm` folgende Syntax bereit:

```
ipvsadm -A -t <vip>[:<port>] -s <scheduler>
```

Die Option `-A` gibt an, dass ein neuer virtueller Dienst erstellt werden soll. Über `-t <vip>[:<port>]` werden die IP-Adresse und gegebenenfalls der Port des Dienstes definiert. Der Scheduling-Algorithmus, mit dem die Anfragen dann verteilt werden, lässt sich mit dem Parameter `-s <scheduler>` angeben.

Zu einem existierenden virtuellen Dienst lassen sich die *Real Server* dann mit

```
ipvsadm -a -t <vip>[:<port>] -r <rip>[:<port>] -m
```

hinzufügen.

Mit der Option `-a` wird ein neuer *Real Server* zur Liste hinzugefügt. Mit `-r <rip>[:<port>]` wird die IP-Adresse und gegebenenfalls der Port des *Real Servers* angegeben, an den die Anfragen weitergeleitet werden. Die Option `-m` gibt an, dass das LVS/NAT Verfahren verwendet werden soll.

Die Weiterleitung für einen Mailserver von der *VIP* `123.123.123.1` zu den *Real Servern* mit den RIPs `192.168.0.101` und `192.168.0.102` lässt sich beispielsweise mit den folgenden Kommandos einrichten:

```
ipvsadm -A -t 123.123.123.1:smtp -s rr
ipvsadm -a -t 123.123.123.1:smtp -r 192.168.0.101:smtp -m -w 1
ipvsadm -a -t 123.123.123.1:smtp -r 192.168.0.102:smtp -m -w 1
```

Als Scheduling-Algorithmus wird hier Round Robin eingesetzt, wobei jeder *Real Server* mit gleicher Gewichtung ausgewählt wird. Durch Weglassen des Dienstes (`:smtp`) in allen drei Kommandos ist es möglich, pauschal alle TCP-Anfragen an die *Real Server* weiter zu reichen, egal an welchem Port sie eintreffen.

Danach sind auf dem Director alle notwendigen Schritte zur Konfiguration von LVS/NAT durchgeführt.

### Konfiguration der Real Server

Auch auf Seite der *Real Server* sind kleine Änderungen notwendig.

So müssen einerseits alle öffentlichen Netzwerkschnittstellen deaktiviert werden, so dass der *Real Server* nur noch über seine private IP-Adresse erreichbar ist.

Außerdem muss dafür gesorgt werden, dass das Standard-Gateway des Real Ser-

vers beim Booten auf die private IP-Adresse des *Directors* gesetzt wird, da keine öffentliche Schnittstelle mehr zur Verfügung steht, über die die Antworten an die Clients geschickt werden können.

Hierfür sorgt folgender Eintrag in der Datei `/etc/conf.d/local.start`:

```
ip route add default via 192.168.0.1
```

Der Umweg über `/etc/conf.d/local.start` ist notwendig, da die private Schnittstelle nicht über `/etc/conf.d/net` konfiguriert wird und somit auch das Setzen des Standard-Gateways in `/etc/conf.d/net` keinen Effekt hat.

### 5.3.3 Konfiguration für LVS/DR

Wie bereits erwähnt sind auch für LVS/DR Anpassungen an den Firewalls aller beteiligten Hosts notwendig (siehe Abschnitt 5.3.1). Abgesehen davon ist die Konfiguration insgesamt einfacher als die von LVS/NAT. Die notwendigen Schritte der Konfiguration umfassen:

**Konfiguration von IPVS:** Auf dem *Director* muss die Weiterleitung ähnlich wie bei LVS/NAT konfiguriert werden.

**Konfiguration der Real Server:** In diesem Schritt muss jedem *Real Server* eine eindeutige öffentliche IP-Adresse zugewiesen werden. Außerdem muss einer Dummy-Schnittstelle die *VIP* des Dienstes zugewiesen werden.

#### Einrichten von IPVS auf dem Director

Die Konfiguration von IPVS erfolgt ähnlich zu LVS/NAT (siehe Abschnitt 5.3.2). Lediglich der Parameter, der die Methode zur Weiterleitung der Anfragen festlegt, ändert sich von `-m` in `-g`. Das Beispiel aus Abschnitt 5.3.2 ändert sich für LVS/DR in:

```
ipvsadm -A -t 123.123.123.1:smtp -s rr
ipvsadm -a -t 123.123.123.1:smtp -r 192.168.0.101:smtp -g -w 1
ipvsadm -a -t 123.123.123.1:smtp -r 192.168.0.102:smtp -g -w 1
```

Firewall-Regeln wie bei LVS/NAT sind nicht notwendig. Auf dem *Director* bleibt dies die einzige erforderliche Änderung. Das Routing an die MAC-Adressen der *Real Server* übernimmt auch hier die IPVS-Software.

#### Konfiguration der Real Server

Auf den *Real Servern* muss zunächst eine öffentliche IP-Adresse konfiguriert werden, über die sie ihre Antworten an die Clients senden.

Zusätzlich muss eine Dummy-Schnittstelle mit der *VIP* konfiguriert werden. Das ist notwendig, damit der *Real Server* erkennt, dass die eintreffenden IP-Pakete für

ihn bestimmt sind. Wichtig ist hier, dass die Dummy-Schnittstelle nicht auf ARP-Anfragen an die *VIP* reagieren darf, da sich ansonsten sowohl *Director* als auch *Real Server* unter der gleichen IP-Adresse melden und eine kontrollierte Weiterleitung der Anfragen durch den *Director* nicht möglich ist. Zusätzlich muss `rp_filter` deaktiviert werden, um die Absendervalidierung (nach RFC1812) zu deaktivieren.

Folgende Kommandos bewirken genau das:

```
/sbin/sysctl -w net.ipv4.conf.all.rp_filter="0"
/sbin/sysctl -w net.ipv4.conf.all.arp_ignore="1"
/sbin/sysctl -w net.ipv4.conf.all.arp_announce="2"
```

Diese Kommandos müssen jedoch ausgeführt werden bevor die Dummy-Schnittstelle gestartet wird, damit diese beim Starten keine ARP-Nachrichten verschickt oder auf ARP-Anfragen antwortet.

In der Netzwerkkonfiguration in Gentoo ist es möglich, vor und nach dem Starten von Netzwerk-Schnittstellen Funktionen aufzurufen.

Mit der folgenden Funktion in `/etc/conf.d/net` ist es möglich die genannten Kommandos immer beim Starten der jeweiligen Schnittstelle auszuführen:

```
preup() {
        if [[ "${IFACE}" == "dummy0" || "${IFACE}" == tunl0 ]]
        then
                /sbin/sysctl -w net.ipv4.conf.all.rp_filter="0"
                /sbin/sysctl -w net.ipv4.conf.all.arp_ignore="1"
                /sbin/sysctl -w net.ipv4.conf.all.arp_announce="2"
        fi
}
```

Das Beispiel einer vollständigen `/etc/conf.d/net` für LVS/DR findet sich im Anhang (siehe Anhang B.9).

### 5.3.4 Konfiguration für LVS/TUN

Die dritte Methode, um die Anfragen an die *Real Server* weiterzuleiten, ist LVS/TUN. Hier werden die eigentlichen IP-Pakete in neue IP-Pakete eingebettet, die an den jeweiligen *Real Server* adressiert sind. Der *Real Server* extrahiert dann das ursprüngliche IP-Paket aus dem umgebenden IP-Frame und verarbeitet dieses.

Damit das funktioniert müssen die *Real Server* die IPIP-Technologie unterstützen. Für Linux basierte *Real Server* bedeutet dies, dass ihr Kernel *IP-Tunneling* unterstützt, oder dass die Unterstützung als Kernel-Modul vorliegt und geladen ist.

Die Konfiguration von LVS/TUN erfolgt analog zu LVS/DR:

**Konfiguration von IPVS:** Auf dem Director muss die IPVS-Weiterleitung konfiguriert werden.

**Konfiguration der Real Server:** Der *Real Server* benötigt wiederum eine eindeu-

tige öffentliche IP-Adresse, sowie eine Tunnel-Schnittstelle, der die virtuelle IP-Adresse des Dienstes (VIP) zugewiesen wird.

#### Einrichten von IPVS auf dem Director

Auch hier unterscheidet sich die Kommandofolge nur durch einen Parameter:

```
ipvsadm -A -t 123.123.123.1:smtp -s rr
ipvsadm -a -t 123.123.123.1:smtp -r 192.168.0.101:smtp -i -w 1
ipvsadm -a -t 123.123.123.1:smtp -r 192.168.0.102:smtp -i -w 1
```

Das Einbetten der IP-Pakete übernimmt IPVS. Auf dem *Director* kann also ein Kernel ohne *IP-Tunneling*-Unterstützung verwendet werden.

#### Konfiguration der Real Server

Die Netzwerkkonfiguration der *Real Server* unterscheidet sich von der bei LVS/DR nur in einem Detail. Statt einer Dummy-Schnittstelle wird eine Tunnel-Schnittstelle benötigt. Diese übernimmt das Extrahieren der IPIP-Pakete (also der eingebetteten IP-Pakete).

Für die Tunnel-Schnittstelle gilt jedoch ebenfalls, dass sie mit der VIP konfiguriert wird und somit jegliche ARP-Kommunikation dieser Schnittstelle unterbunden werden muss. Die Art und Weise wie dies geschieht ist identisch zu LVS/DR.

Für LVS/TUN findet sich ebenfalls ein Beispiel für eine `/etc/conf.d/net`-Datei im Anhang (siehe Anhang B.10).

Damit LVS/TUN funktioniert, müssen die *Real Server* die IPIP-Technik unterstützen. Ist dies der Fall, können die Anfragen auch über Netzwerkgrenzen hinweg an geografisch verteilte Computer geroutet werden, solange deren IP-Adressen aufgelöst werden können.

### 5.3.5 Bewertung der Methoden zur Weiterleitung

Die Lastverteilung über LVS/NAT stellt wohl die sicherste Möglichkeit dar, da sich die *Real Server* bis auf die tatsächlich benötigten Ports vom Internet isolieren lassen. Auch werden hierfür keine zusätzlichen IP-Adressen benötigt, was bei LVS/DR und LVS/TUN in der vorgegebenen Umgebung der Fall wäre, da diese hier für den Rückkanal eine direkte Internetanbindung benötigen. Die Vorteile von LVS/DR und LVS/TUN liegen vor allem in der einfachen Konfiguration. LVS/DR unterliegt allerdings der Einschränkung, dass es nur funktioniert, wenn die beteiligten Maschinen sich im gleichen Netzwerksegment befinden, da ansonsten die Pakete nicht vom *Director* zu den *Real Servern* geroutet werden können. Mit der Möglichkeit zur geografischen Verteilung der *Real Server* ist LVS/TUN interessant für hohe Ansprüche an Hochverfügbarkeit, da dadurch unter Anderem auch Naturkatastrophen als Ausfallursache eliminiert werden können.

LVS/NAT und LVS/DR sind auch plattformübergreifend einsetzbar, da die *Real Server* die eintreffenden Anfragen behandeln können, als kämen sie direkt vom Client. Für LVS/TUN gilt dies nicht. Hier muß der *Real Server* die IPIP-Technologie unterstützen, um die eingebetteten Nachrichten extrahieren zu können. Während beispielsweise Windows 2000 noch IPIP unterstützt, können Maschinen mit Windows Server 2003 nicht mehr mit IPIP-Paketen umgehen.

Für den *Director* wird auf jeden Fall ein Linux-System benötigt, da IPVS fester Bestandteil des Linux-Kernels ist.

### 5.3.6 Failover für LVS mit Keepalived

Damit ein LVS-Cluster auch auf Ausfälle einzelner Dienste oder auch ganzer Maschinen reagieren kann, wird eine zusätzliche Software benötigt, die einerseits die Ausfälle erkennt und andererseits die notwendigen Modifikationen an den Loadbalancern durchführt. Keepalived implementiert das Virtual Router Redundancy Protocol (VRRP). Dieses Protokoll wurde mit dem Ziel entwickelt, die Ausfallsicherheit von Routern zu erhöhen. Die Router werden zu Gruppen zusammengefasst, von denen jede eine ID erhält. Ein Router aus der Gruppe wird Master. Die restlichen Router sind Slaves. Der Master sendet in regelmäßigen Abständen Nachrichten an die speziell für diesen Zweck reservierte Multicast Adresse `224.0.0.18`. Die Slaves erkennen anhand dieser Nachrichten, ob der Master noch aktiv ist und können bei einem Ausfall des Masters entsprechend reagieren. Damit Keepalived korrekt arbeitet, muss eventuell diese Adresse an den Firewalls freigeschaltet werden (dies ist in der Modifikation in Anhang B.7 bereits vorgesehen).

### 5.3.7 Installation von Keepalived

Während des Kompilierens von Keepalived müssen unter `/usr/src/linux` die Kernel-Quellen mit aktiviertem IPVS-Code liegen, da Keepalived ansonsten ohne Unterstützung für IPVS gebaut wird. Keepalived könnte in diesem Fall die notwendigen Änderungen am IPVS-Loadbalancing nicht vornehmen.

### 5.3.8 Konfiguration von Keepalived

Die komplette Konfiguration eines hochverfügbaren LVS-Clusters erfolgt in der Konfigurationsdatei `/etc/keepalived/keepalived.conf` auf den *Directors*.

Diese Konfigurationsdatei besteht im Wesentlichen aus drei Teilen. Ein Teil mit globalen Parametern, ein Teil für die VRRP-Parameter und ein Teil, in dem LVS konfiguriert wird.

### Globale Parameter

Im Abschnitt `global_defs` der Konfigurationsdatei können globale Parameter eingestellt werden. Diese umfassen die `router_id`, die die Keepalived-Instanz eindeutig identifiziert, sowie alle notwendigen Mailserver-Parameter um Benachrichtigungen per E-Mail versenden zu können.

```
global_defs {
        notification_email {
                webmaster@your.domain.name
        }
        notification_email_from director1@your.domain.name
        smtp_server 192.168.0.101
        smtp_connect_timeout 30
        router_id director1
}
```

Diese Konfiguration kann für alle Keepalive-Instanzen übernommen werden. Lediglich die `router_id` muss für jede Instanz abgeändert werden.

### VRRP Parameter

Um das VRRP-Verhalten der Keepalived-Instanz zu konfigurieren existieren die beiden Abschnitte `vrrp_instance` und `vrrp_sync_group`.

Im Abschnitt `vrrp_instance` wird definiert, ob es sich um eine Master- oder Slave-Instanz handelt, über welche Netzwerkschnittstelle die VRRP-Nachrichten verschickt werden und die `virtual_router_id`, die für alle Master- und Slave-Instanzen einer VRRP-Instanz gleich sein muss.
Durch die Verwendung unterschiedlicher `virtual_router_id`s können bis zu 255 VRRP-Instanzen gleichzeitig betrieben werden. Mit einem `virtual_ipaddress`-Block lassen sich dann die IP-Adressen konfigurieren, die an diese VRRP-Instanz gebunden sind. Fällt der Master aus, übernimmt ein Slave alle IP-Adressen, die in diesem Block konfiguriert werden. Über `priority` ist es möglich, die Reihenfolge zu konfigurieren, in welcher beim Failover bzw. Failback die Slave-Instanzen die VRRP-Instanz übernehmen. In einem Setup mit drei Routern (ein Master und zwei Slaves) würde standardmäßig der Master alle Anfragen bearbeiten und die virtuelle IP-Adresse an eine seiner Netzwerkschnittstellen binden. Fällt dieser Master aus, übernimmt der Slave, in dessen Konfiguration der höhere `priority`-Wert eingestellt ist. Fällt auch dieser Router aus, so übernimmt der zweite Slave. Desweiteren können in diesem Abschnitt Authentifizierungseinstellungen, das Intervall, in dem VRRP-Nachrichten verschickt werden sollen, sowie gegebenenfalls ein Standardgateway für die VRRP-Instanz konfiguriert werden.

Mit Hilfe des `vrrp_sync_group`-Blocks lassen sich mehrere VRRP-Instanzen zu Gruppen zusammenfassen. Im Fehlerfall werden alle IP-Adressen dieser Gruppe auf einen Slave umgezogen, sobald eine VRRP-Instanz der Gruppe als ausgefallen an-

genommen wird.

Ein Beispiel für die VRRP-Konfiguration eines hochverfügbaren LVS/NAT-Setups, bei dem sich sowohl die virtuelle IP (`123.123.123.1`) als auch die private Adresse des NAT-Gateways (`192.168.0.1`) auf dem selben *LVS-Director* befinden müssen, könnte dann wie folgt aussehen:

```
vrrp_sync_group mailserver {
        group {
                vip
                gwip
        }
}

vrrp_instance vip {
        state MASTER
        interface eth0
        virtual_router_id 50
        priority 150
        advert_int 1
        authentication {
                auth_type PASS
                auth_pass topSecret
        }
        virtual_ipaddress {
                123.123.123.1
        }
}

vrrp_instance gwip {
        state MASTER
        interface eth1
        virtual_router_id 51
        priority 150
        advert_int 1
        authentication {
                auth_type PASS
                auth_pass verySecret
        }
        virtual_ipaddress {
                192.168.0.1
        }
}
```

Auf den Slaves müssen jeweils `state` und `priority` für jede `vrrp_instance` angepasst werden.

#### LVS Parameter

Dieser Abschnitt enthält für jeden virtuellen Dienst, der über LVS verteilt wird, einen `virtual_server`-Block, in dem über die Optionen `lb_algo` und `lb_kind` der Scheduling-Algorithmus und die Weiterleitungsmethode (NAT, TUN oder DR) für jeden virtuellen Dienst konfiguriert werden können. `persistence_timeout` gibt an, wie lange persistente Verbindungen gespeichert werden. Innerhalb dieses Timeouts werden alle Anfragen eines Clients an den selben *Real Server* weitergeleitet. Über eingebettete `real_server`-Blöcke werden die *Real Server* definiert. Für jeden *Real Server* lässt sich die Gewichtung, sowie der Healthchecker konfigurieren. Als Healthchecker unterstützt Keepalived mit `HTTP_GET` und `SSL_GET` bereits Überprüfungsmechanismen für Webserver. Mit `SMTP_CHECK` lassen sich auch Mailserver auf ihre korrekte Funktionalität überprüfen. Alle anderen Dienste können, mit Hilfe von `MISC_CHECK`, durch externe Programme und Skripten oder direkt (mit `TCP_CHECK`) überprüft werden.

Ein zu den beiden vorangehenden Beispielen passender LVS-Abschnitt, über den zwei *Real Server* bedient und überprüft werden, könnte demnach folgendermaßen aussehen:

```
virtual_server 123.123.123.1 25 {
        delay_loop 6
        lb_algo rr
        lb_kind NAT
        persistence_timeout 50
        protocol TCP

        real_server 192.168.178.0.101 25 {
                weight 1
                SMTP_CHECK {
                        connect_timeout 3
                }
        }

        real_server 192.168.178.0.102 25 {
                weight 1
                SMTP_CHECK {
                        connect_timeout 3
                }
        }
}
```

### 5.3.9 LVS-Setup für Mail- und Webserver

Um einen Mailserver mit Hilfe von LVS hochverfügbar zu machen, benötigt man einerseits redundante *Real Server*, um den Ausfall eines *Real Servers* abfangen zu

können, sowie zwei *LVS-Directors*, um auch hier beim Ausfall eines *Directors* auf ein Backup-System umschalten zu können.

Um auch die Ressourcen des Backup-*Directors* sinnvoll zu nutzen, bietet es sich an, einen weiteren Dienst über diesen *Director* zu verteilen und dadurch hochverfügbar zu machen.

Ein Szenario, bei dem ein *Director* die Mailserver und der zweite *Director* redundante Webserver bedient, bietet sich also an (wie zu Beginn von Abschnitt 5.3 skizziert). Fällt einer der beiden *Directors* aus, übernimmt der zweite dessen IP-Adressen und leitet auch die Anfragen an diese IP-Adressen an die *Real Server* weiter (dieses Setup entspricht dem, das bereits in Abbildung 5.1 skizziert wurde).

Für ein derartiges Setup findet sich in Anhang B.11 ein Konfigurationsbeispiel.

## 5.4 Loadsharing mit ClusterIP

In Clustern, die die ClusterIP-Technik einsetzen, wird ähnlich wie beim Loadbalancing mit LVS die Last der Anfragen auf die Clusterknoten verteilt.

Allerdings erfolgt die Verteilung der Last nicht zentral durch die Loadbalancer. Die Clusterknoten entscheiden anhand eines Hashing-Algorithmus selbst, welche Anfragen sie bearbeiten und für welche Anfragen sie nicht zuständig sind.

Aufgrund dieses Unterschiedes wird diese Methode auch als *Loadsharing* bezeichnet (siehe [11]).

### 5.4.1 Installation und Konfiguration

#### Installation von ClusterIP

Für die Installation über das Portage-System von Gentoo müssen unter `/usr/src/linux` die Kernel Quellen liegen. Ist dies der Fall, so kann `iptables` mit dem notwendigen `extensions`-USE-Flag übersetzt werden.

Mit iptables und einem Kernel mit aktiviertem ClusterIP-Code erfüllt das System alle Vorraussetzungen für einen ClusterIP Knoten.

#### Konfiguration von ClusterIP

Damit alle Cluster-Knoten auf die gemeinsame IP-Adresse reagieren, muß die IP-Adresse zunächst auf allen Knoten aktiviert werden. Dies erfolgt beispielsweise mit:

```
ip addr add 123.123.123.1 dev eth0
```

Danach kann `CLUSTERIP` als Target für `iptables`-Filterregeln verwendet werden. Für das `CLUSTERIP`-Target stehen dann zusätzliche `iptables`-Optionen zur Verfügung:

`--new`: Kennzeichnet eine neue ClusterIP

`--hashmode`: Gibt die Hashing-Methode an, anhand welcher die Cluster-Knoten entscheiden, wer für die Anfrage zuständig ist. Mögliche Argumente sind `sourceip`, `sourceip-sourceport` und `sourceip-sourceport-destport`.

`--clustermac`: Gibt die Multicast-MAC-Adresse an, der die ClusterIP zugeordnet wird.

`--total-nodes`: Die Anzahl der Knoten im Cluster.

`--local-node`: Nummer des Knotens.

`--hash-init`: Initialisierungswert für das Hashing.

Während der Tests führte das Entfernen der iptables-Regel, die das `CLUSTERIP`-Target definiert, reproduzierbar zu Kernel-Paniken, die einen kompletten Absturz der U-Domäne nach sich zogen. Für den produktiven Einsatz in der Xen-Umgebung ist ClusterIP deshalb noch nicht geeignet.

### 5.4.2 Failover

#### Notwendige Schritte für einen Failover

In einem fertig konfigurierten Cluster muss ein anderer Cluster-Knoten zusätzlich die Anfragen des ausgefallenen Knotens bearbeiten. ClusterIP stellt über das Proc-Dateisystem eine Schnittstelle bereit, um nachträglich die Hashwerte zu ändern, auf die der Knoten reagiert.

Das Kommando

```
echo "+2" > /proc/net/ipt_CLUSTERIP/123.123.123.1
```

weist ClusterIP an, zusätzlich alle Anfragen mit einem Hashwert von `2` zu beantworten.

Fällt der Cluster-Knoten, der mit `--local-node 2` konfiguriert wurde aus, kann ein anderer Knoten mit diesem Kommando einen Failover durchführen.

Ist der ausgefallene Knoten wieder einsatzbereit, erfolgt ein Failback mit dem entsprechenden Kommando:

```
echo "-2" > /proc/net/ipt_CLUSTERIP/123.123.123.1
```

Damit im Fehlerfall der Failover mit anschließendem Failback automatisch durchgeführt wird, wird wieder eine zusätzliche Software benötigt, die den Ausfall eines Knotens erkennt und die notwendigen Schritte durchführt. Eine Möglichkeit hierfür ist Heartbeat aus Abschnitt 5.2.

### Automatischer Failover mit Heartbeat

Für Heartbeat existiert bereits ein erweiterter `IPaddr2` *Resource-Agent*, der Failover für ClusterIP-Systeme ermöglicht. Seit der Heartbeat Version 2.1.1 ist dieser Agent fester Bestandteil des Heartbeat-Pakets.

Neben IP-Adresse und Netzwerk-Schnittstelle können dem *Resource-Agent* die Hashing-Methode und die MAC-Adresse für den ClusterIP-Mechanismus übergeben werden.

Damit mehrere Instanzen dieser Ressource im Cluster auftreten können, muss sie als Clone-Ressource konfiguriert werden. Über den Parameter `clone_max` wird konfiguriert, wie viele Instanzen der Ressource maximal auftreten dürfen. Für ClusterIP sollte der Wert dieses Parameters gleich der Anzahl der Knoten im Loadsharing-Cluster sein, damit jeder Knoten auch tatsächlich am Loadsharing teilnimmt.

Zusätzlich muss über das Attribut `clone_node_max` die maximale Anzahl der Instanzen angegeben werden, die auf einem Knoten laufen dürfen. Für den Extremfall, dass alle Knoten bis auf einen ausfallen, muß der letzte Knoten im Cluster die Ressourcen aller anderen Knoten übernehmen, damit keine Anfragen unbeantwortet bleiben. Deshalb sollte der Wert von `clone_node_max` gleich dem von `clone_max` gewählt werden.

Fertig konfiguriert übernimmt der *Resource-Agent* dann alle notwendigen Schritte zur Konfiguration der ClusterIP-Knoten und im Fehlerfall führt er die Failover- und Failback-Schritte aus Abschnitt 5.4.2 automatisch aus.

In Anhang B.12 findet sich ein Beispiel für eine Heartbeat-Konfiguration eines Loadsharing-Clusters, der aus zwei ClusterIP-Knoten besteht.

# 6 Zusammenfassung und Ausblick

Die im vorangegangenen Abschnitt vorgestellten Lösungen behandeln mit Linux-HA die bekannteste Open Source Hochverfügbarkeitslösung und mit Linux Virtual Server und ClusterIP zwei unterschiedliche Mechanismen, die zusätzlich zum Gewinn an Ausfallsicherheit auch eine Lastverteilung der Dienste ermöglichen.

Linux-HA ist vergleichbar mit kommerziellen HA-Produkten und eignet sich vor allem für den Aufbau von Aktiv/Passiv-Clustern. Für Dienste, die nicht *Cluster-Aware* sind, stellt eine derartige HA-Lösung die einzige Möglichkeit dar, die Verfügbarkeit zu erhöhen. Ein Fileserver, der ein Dateisystem freigibt, das nicht clusterfähig ist, kann nicht mit Hilfe eines Aktiv/Aktiv-Clusters abgesichert werden, da immer nur ein Knoten das Dateisystem mounten darf. Clusterfähige Dienste, wie einfache Webserver, können durch Aktiv/Passiv-Systeme, aber zusätzlich auch durch mehrere aktive Knoten, die die Last unter sich aufteilen, hochverfügbar gemacht werden.

Da sich die ClusterIP-Lösung in den Tests aufgrund der Systemabstürze für unsere spezielle, virtualisierte Systemumgebung nicht bewährt hat, bleibt in unserem Serverpool Linux Virtual Server die einzige Lösung um Aktiv/Aktiv-Cluster zu erstellen. Zusätzlich zu den mindestens zwei Cluster-Knoten benötigt ein solches System allerdings noch mindestens zwei Loadbalancer, um sowohl die Server als auch die Loadbalancer ausfallsicher zu machen.

Insgesamt erhöht sich durch den Einsatz von HA-Software die Komplexität der IT-Systeme, was zusätzlichen Konfigurations- und Wartungsaufwand mit sich bringt. Wie an der Umsetzung des relativ einfachen Beispiels eines Mailserversystems in den vorausgegangenen Abschnitten sichtbar wird, bringt die Einführung einer HA-Struktur in eine bestehende (oder auch neu zu entwickelnde) Serverumgebung eine weitere Ebene in der Architektur des Systems mit sich, die bei der Konzeption mit modelliert werden muss. Diese Aufgabe erfordert auch von den mit der Konfiguration und Pflege einer Systemumgebung betrauten IT-Mitarbeitern zusätzliche Qualifikationen durch entsprechende Schulungsmaßnahmen. Wird dafür nicht ausreichend Sorge getragen, so ist bei eventuell resultierenden fehlerhaften Systemkonfigurationen oder nicht sachgemäßer Wartung damit zu rechnen, dass sich die Ausfallwahrscheinlichkeit des Systems insgesamt unter Umständen sogar erhöhen kann.

Auf dem Gebiet der Hochverfügbarkeit sind vor allem für Virtualisierungslösungen in Zukunft viele Neuerungen zu erwarten. Da virtualisierte Systeme unabhängig von der Hardware des Hostsystems sind, können sie ohne großen Aufwand auf andere Hosts migriert werden. Teilweise ist dies sogar im laufenden Betrieb möglich. Durch die Integration von Monitoring- und Failover-Komponenten in die Virtualisierungssysteme wird zusätzliche Failover-Software überflüssig, was die Komplexität der Systeme reduziert.

# A Wichtige Begriffe

**0-Domäne:** Die privilegierte Xen-Domäne, die direkten Zugriff auf die Hardware hat. Alle U-Domänen werden von einer 0-Domäne aus gestartet.

**Aktiv/Aktiv-Cluster:** Ein Cluster, in dem mehrere Knoten gleichzeitig Dienste bereit stellen. Fällt ein aktiver Knoten aus, stellt immer noch mindestens ein weiterer aktiver Knoten die hochverfügbaren Dienste bereit.

**Aktiv/Passiv-Cluster:** Ein Hochverfügbarkeitscluster, in dem nur der aktive Knoten Dienste bereit stellt. Alle anderen Knoten im Cluster sind passiv. Sie stellen keine Dienste zur Verfügung, solange der aktive Knoten verfügbar ist. Ist der aktive Knoten nicht mehr verfügbar, übernimmt ein passiver Knoten seine Dienste und wird somit selbst aktiv.

**Cluster:** Ein Verbund von Computern um die Rechenkapazität oder die Verfügbarkeit zu erhöhen (siehe [24]).

**Direct Routing:** Eine Methode zur Weiterleitung von IP-Paketen, bei der das Gateway die eintreffenden Pakete direkt an die Ethernet-Adresse (MAC-Adresse) des Zielrechners weiterleitet.

**Director:** Als Director wird bei IPVS ein Loadbalancer bezeichnet.

**Downtime:** Die Zeit in der ein System aufgrund von Wartungsarbeiten, Defekten oder anderen Zwischenfällen nicht erreichbar ist.

**Failback:** Der Wechsel vom Standby-System zum Primärserver, nachdem dieser wieder voll funktionsfähig ist.

**Failover:** Der ungeplante Wechsel von einem Primärserver zu einem Standby-System (siehe [25]).

**Floating IP:** Beim Failover einer IP-Adresse von einem Knoten auf einen anderen wird die IP-Adresse, die beim Failover übernommen wird, als Floating IP bezeichnet.

**HA:** kurz für High Availability.

**Heartbeat:** Ein „Lebenszeichen“, das ein Knoten im Cluster verschickt, um alle Cluster-Partner darüber zu informieren, dass der Knoten noch voll funktionsfähig ist.

**High Availability:** engl. für Hochverfügbarkeit.

**Hochverfügbarkeit:** Ein System gilt als hochverfügbar, wenn eine Anwendung auch im Fehlerfall weiterhin verfügbar ist und ohne unmittelbaren menschlichen Eingriff weiter genutzt werden kann (siehe [26]).

**IP-Tunneling:** Eine Methode zur Weiterleitung von IP-Paketen, bei der das Gateway die eintreffenden Pakete in andere IP-Pakete einbettet, die an den tatsächlichen Empfänger adressiert sind.

**Knoten:** Ein einzelner Computer in einem Cluster.

**Loadbalancing:** Alle Anfragen werden anhand eines Scheduling-Algorithmus' auf mehrere redundante Systeme verteilt.

**NAT:** kurz für Network Address Translation. Eine Methode zur Weiterleitung von IP-Paketen, bei der das Gateway die Zieladressen der eintreffenden Pakete entsprechend der privaten IP-Adresse des Zielrechners ändert.

**NIC:** kurz für Network Interface Card (Netzwerkkarte).

**Real Server:** Die Server, an die IPVS die Anfragen verteilt. Also beispielsweise die eigentlichen Mail- und Web-Server, die die Anfragen der Clients bearbeiten und darauf antworten.

**Resource-Agent:** Als Resource-Agent wird bei Heartbeat ein Skript bezeichnet, mit dessen Hilfe sich die Ressourcen (die Dienste) einer HA-Instanz steuern lassen.

**Shared Storage:** Ein gemeinsam genutzter Festspeicher (z.B. ein Disk Array), auf den von mehreren Rechnern über ein Netzwerk gemeinsam und eventuell auch gleichzeitig (konkurrierend) zugegriffen werden kann (siehe [27]).

**Single Point of Failure:** Eine Komponente eines Systems, deren Ausfall zum Ausfall des Gesamtsystems führt.

**SPOF:** kurz für Single Point of Failure.

**STONITH:** kurz für „Shoot the other node in the head".

**U-Domäne:** Eine unprivilegierte virtualisierte Xen-Domäne.

**VIP:** Die IP-Adresse, unter der der eigentliche Dienst eines Linux Virtual Server-Clusters erreicht werden kann. Der Client richtet seine Anfragen immer an diese IP-Adresse. Der Director leitet diese Anfrage dann an die *Real Server* weiter.

**VRRP:** Das Virtual Router Redundancy Protokoll wurde mit der Absicht entwickelt, einen Standard zu schaffen, der die Ausfallsicherheit von Routern erhöhen soll. Mehrere Router werden hier zu einer Gruppe zusammengefasst. Einer der Router wird zum Master und erhält eine MAC- und eine IP-Adresse. Der Master sendet in regelmäßigen Abständen Heartbeats an die IP-Adresse `224.0.0.18` (siehe [20]).

# B Konfigurationsdateien

## B.1 Grundkonfiguration von Heartbeat

### B.1.1 /etc/ha.d/ha.cf

In der Datei `/etc/ha.d/ha.cf` werden vor allem die Netzwerkparameter für die Heartbeat-Verbindung konfiguriert.

```
crm on
use_logd on

node node-1 node-2

ucast eth1 192.168.0.50
ucast eth1 192.168.0.51

udpport 694

keepalive 100ms
deadtime 1
initdead 10
```

Dabei bedeuten die einzelnen Parameter:

`crm on`: Den *Cluster Resource Manager* verwenden, um alle neuen Features von Heartbeat 2 verwenden zu können. Bleibt der CRM deaktiviert, so wird Heartbeat im zur Version 1 kompatiblen Modus betrieben.

`use_logd on`: Den Log-Dämon verwenden, um vom eventuell blockierenden Syslog-Dämon entkoppelt zu sein.

`node`: Alle Knoten, die zum Cluster gehören sollen. Für jeden Knoten muß hier sein Hostname stehen, wie ihn `uname -n` liefert.

`ucast eth1 192.168.0.50`: Die Zieladressen, an die Heartbeat-Pakete geschickt werden, sowie die Netzwerkschnittstelle, über die die Pakete verschickt werden.

`udpport 694`: Der UDP-Port, auf dem Heartbeat auf eingehende Nachrichten anderer Clusterknoten wartet.

`keepalive 100ms`: Das Intervall, in dem die Heartbeats verschickt, und in dem die Heartbeats aller anderen Clusterknoten erwartet werden.

`deadtime 1`: Die Zeit (hier in Sekunden), nach der ein Knoten als tot angenommen wird, wenn für diese Dauer keine Heartbeats des Knotens eintreffen. Für den aktuellen Wert (also eine Sekunde) dürfen also bis zu neun Heartbeats verloren gehen, ohne dass der Knoten der diese verschickt fälschlicherweise als tot angenommen wird, da alle 100 Millisekunden ein Heartbeat verschickt wird.

`initdead 10`: Zeit die vergehen muss, nachdem Heartbeat gestartet wurde, ehe der Knoten andere als tot annimmt.

### B.1.2 /etc/ha.d/ha_logd.cf

Der Log-Dämon, der Heartbeat vom herkömmlichen Syslog-Dämon entkoppelt, wird über die Datei `/etc/ha.d/ha_logd.cf` konfiguriert. Die Standardkonfiguration sollte hier ausreichend sein:

```
debugfile  /var/log/ha-debug
logfile    /var/log/ha-log
```

Hier werden die Dateien angegeben, in welche die Debug- und Lognachrichten geschrieben werden sollen.

### B.1.3 /etc/ha.d/authkeys

Die `authkeys`-Datei enthält ein Passwort, mit dem sich die Clusterknoten gegenseitig authentifizieren.

```
auth 1
1 sha1 StrengGeheimePassphrase
```

`auth 1`: Gibt an, dass ausgehende Nachrichten mit dem ersten Schlüssel signiert werden sollen.

`1 sha1 Passphrase`: Gibt den Schlüssel und das Verfahren an, mit dem die Nachrichten signiert werden sollen.

## B.2 IPaddr Resource Agent

Ein „Diff“ auf die Original-Version und die modifizierte Version des *Resource-Agent* ergibt folgende Änderungen:

```
/usr/lib/ocf/resource.d # diff myResources/IPaddr heartbeat/IPaddr
20d19
< #     OCF_RESKEY_gateway
462,476d460
< add_route () {
<   iface_base="$1"
```

```
<   gateway="$2"
<
<   case "$SYSTYPE" in
<       SunOS)  CMD="";;
<       *BSD)   CMD="";;
<       *)      CMD="$ROUTE add default gw $gateway $iface_base";;
<   esac
<
<   $CMD
<
<   return $?
< }
<
513,515c497
< #       CMD="$IFCONFIG $iface $ipaddr netmask $netmask \
        broadcast $broadcast";;
<         # Maybe we have to bring up the Interface before the alias
<         CMD="$IFCONFIG $iface_base up && $IFCONFIG $iface \
        $ipaddr netmask $netmask broadcast $broadcast";;
---
>         CMD="$IFCONFIG $iface $ipaddr netmask $netmask \
        broadcast $broadcast";;
619,623d600
<     # Set the Default Route if necessary
<     if [ -n "$OCF_RESKEY_gateway" ]; then
<       add_route "$OCF_RESKEY_nic" "$OCF_RESKEY_gateway"
<     fi
<
```

## B.3 Gentoo Service Resource-Agent

Ein OCF-konformes Wrapper-Skript, das Dienste über die Gentoo-Init-Skripten startet, und sich dabei an die von Heartbeat geforderten Konventionen hält:

```
#!/bin/bash
#
#       Resource Agent for Gentoo Services
#
#       usage: $0  {start|stop|monitor|meta-data|validate-all}
#
##############################
# Initialization:

. /usr/lib/heartbeat/ocf-shellfuncs
```

```
############################

Postfix_Metadata() {
        cat <<END
<?xml version="1.0"?>
<!DOCTYPE resource-agent SYSTEM "ra-api-1.dtd">
<resource-agent name="GentooService">
<version>1.0</version>

<longdesc lang="en">
Resource Agent to start, stop an monitor Gentoo-Services.
</longdesc>
<shortdesc lang="en">Gentoo-Service resource agent</shortdesc>

<parameters>

</parameters>

<actions>
<action name="start" timeout="30" />
<action name="stop" timeout="30" />
<action name="monitor" depth="0" timeout="30" interval="10" \
        start-delay="30" />
<action name="meta-data" timeout="5" />
<action name="validate-all" timeout="5" />
</actions>
</resource-agent>
END
}

Monitor() {
        /etc/init.d/$OCF_RESKEY_service --quiet status
        rc=$?
        if
                [ $rc -eq 0 ]
        then
                ocf_log info "$OCF_RESKEY_service is running"
                return $OCF_SUCCESS
        else
                ocf_log info "$OCF_RESKEY_service is stopped"
                return $OCF_NOT_RUNNING
        fi
}

Start() {
        Monitor
```

```
        monitor=$?
        if [ $monitor = $OCF_SUCCESS ]
        then
                ocf_log info "$OCF_RESKEY_service already running."
                return $OCF_SUCCESS
        else
                ocf_log info "Starting $OCF_RESKEY_service ..."
                /etc/init.d/$OCF_RESKEY_service --quiet start
                started=$?
                if [ $started -ne 0 ]
                then
                        return $OCF_ERR_PERM
                else
                        return $OCF_SUCCESS
                fi
        fi
}

Stop() {
        Monitor
        monitor=$?
        if [ $monitor = $OCF_NOT_RUNNING ]
        then
                ocf_log info "$OCF_RESKEY_service already stopped."
                return $OCF_SUCCESS
        else
                ocf_log info "Stopping $OCF_RESKEY_service ..."
                /etc/init.d/$OCF_RESKEY_service --quiet stop
                stopped=$?
                if [ $rc -ne 0 ]
                then
                        return $OCF_ERR_PERM
                else
                        return $OCF_SUCCESS
                fi
        fi
}

Validate_All() {
  return $OCF_SUCCESS
}

if [ $# -ne 1 ]
then
  exit $OCF_ERR_ARGS
fi
```

```
case $1 in
  meta-data)            Metadata
                        exit $OCF_SUCCESS
                        ;;
  start)                Start
                        ;;
  stop)                 Stop
                        ;;
  monitor)              Monitor
                        ;;
  validate-all)         Validate_All
                        ;;
esac
exit $?
```

Über einen Parameter wird dem *Resource-Agent* der Name des Init-Skriptes übergeben, das Heartbeat starten, stoppen und überwachen soll. Mit Hilfe dieses *Resource-Agents* können somit auch Dienste wie Postfix, UW-IMAP und Apache von Heartbeat gesteuert werden.

## B.4 Filesystem Resource-Agent

Änderungen am Filesystem *Resource-Agent*, um einen Lazy-Unmount durchzuführen, falls der erste Unmount-Versuch nicht erfolgreich war:

```
558a559,560
>
>     lazy_umount="-l"
584c586,591
<               ocf_log err "Couldn't unmount $SUB, giving up!"
---
>               ocf_log err "Couldn't unmount $SUB, doing \
        lazy unmount!"
>       if $UMOUNT $lazy_umount $SUB ; then
>         rc=$OCF_SUCCESS
>       else
>         ocf_log err "Lazy unmount failed!"
>       fi
```

## B.5 Beispiel für die CIB eines Mail-Servers

Beispiel der XML-Definition der Ressourcen eines Mail-Servers als Ressourcengruppe:

```
<resources>
<group id="mail-server">
 <primitive class="ocf" id="mail-homes" provider="myResources" \
        type="Filesystem" is_managed="true">
  <operations>
   <op id="homes_monitor" name="monitor" interval="5s"/>
  </operations>

  <instance_attributes id="mail-homes-attributes">
   <attributes>
    <nvpair id="home_device" name="device" \
        value="192.168.0.3:/export/mail/home"/>
    <nvpair id="home_directory" name="directory" value="/home"/>
    <nvpair id="home_fstype" name="fstype" value="nfs"/>
    <nvpair id="home_options" name="options" \
        value="rw,tcp,rsize=32768,wsize=32768"/>
   </attributes>
  </instance_attributes>
 </primitive>

 <primitive class="ocf" id="mail-spool" provider="myResources" \
        type="Filesystem" is_managed="true">
  <operations>
   <op id="spool_monitor" name="monitor" interval="5s"/>
  </operations>
  <instance_attributes id="mail-spool-attributes">
   <attributes>
    <nvpair id="spool_device" name="device" \
        value="192.168.0.3:/export/mail/spool"/>
    <nvpair id="spool_directory" name="directory" \
        value="/var/spool/mail"/>
    <nvpair id="spool_fstype" name="fstype" value="nfs"/>
    <nvpair id="spool_options" name="options" \
        value="rw,tcp,rsize=32768,wsize=32768"/>
   </attributes>
  </instance_attributes>
 </primitive>

 <primitive class="ocf" id="postfix" provider="myResources" \
        type="GentooService" is_managed="true">
  <operations>
```

```
   <op id="postfix_monitor" name="monitor" interval="5s"/>
  </operations>
  <instance_attributes id="postfix-attributes">
   <attributes>
    <nvpair id="postfix_service" name="service" value="postfix"/>
   </attributes>
  </instance_attributes>
 </primitive>

 <primitive class="ocf" id="xinetd" provider="myResources" \
        type="GentooService" is_managed="true">
  <operations>
   <op id="xinetd_monitor" name="monitor" interval="5s"/>
  </operations>
  <instance_attributes id="xinetd-attributes">
   <attributes>
    <nvpair id="xinetd_service" name="service" value="xinetd"/>
   </attributes>
  </instance_attributes>
 </primitive>

 <primitive class="ocf" id="private-net" provider="myResources" \
        type="IPaddr" is_managed="true">
  <instance_attributes id="srv_attributes">
   <attributes>
    <nvpair id="priv_ip" name="ip" value="192.168.0.1"/>
    <nvpair id="priv_mask" name="netmask" value="24"/>
    <nvpair id="priv_nic" name="nic" value="eth1"/>
   </attributes>
  </instance_attributes>
 </primitive>

 <primitive class="ocf" id="public" provider="myResources" \
        type="IPaddr" is_managed="true">
  <instance_attributes id="sb21_ip_attributes">
   <attributes>
    <nvpair id="pub_ip" name="ip" value="123.123.123.1"/>
    <nvpair id="pub_mask" name="netmask" value="24"/>
    <nvpair id="pub_nic" name="nic" value="eth0"/>
    <nvpair id="pub_gw" name="gateway" value="123.123.123.254"/>
   </attributes>
  </instance_attributes>
 </primitive>
</group>
</resources>
```

## B.6 Beispiel für die CIB eines NFS-Servers

Auch für die beiden NFS-Server können *Resource-Agents* verwendet werden. Das Mounten des Speichersystems übernimmt dann wieder der `Filesystem`-*Resource-Agent*, und das Starten und Stoppen der Dienste der `GentooService`-*Resource-Agent*.

```
<resources>
<group id="nfs-server">
 <primitive class="ocf" id="nfs-exports" provider="myResources" \
        type="Filesystem" is_managed="true">
  <operations>
   <op id="exports_monitor" name="monitor" interval="5s"/>
  </operations>

  <instance_attributes id="exports-attributes">
   <attributes>
    <nvpair id="exports_device" name="device" \
        value="/dev/hda"/>
    <nvpair id="exports_directory" name="directory" \
        value="/export"/>
    <nvpair id="exports_fstype" name="fstype" value="ext3"/>
   </attributes>
  </instance_attributes>
 </primitive>

 <primitive class="ocf" id="nfs" provider="myResources" \
        type="GentooService" is_managed="true">
  <operations>
   <op id="nfs_monitor" name="monitor" interval="5s"/>
  </operations>
  <instance_attributes id="nfs-attributes">
   <attributes>
    <nvpair id="nfs_service" name="service" value="nfs"/>
   </attributes>
  </instance_attributes>
 </primitive>

 <primitive class="ocf" id="private-net" provider="myResources" \
        type="IPaddr" is_managed="true">
  <instance_attributes id="priv_attributes">
   <attributes>
    <nvpair id="priv_ip" name="ip" value="192.168.0.3"/>
    <nvpair id="priv_mask" name="netmask" value="24"/>
    <nvpair id="priv_nic" name="nic" value="eth1"/>
   </attributes>
```

```
  </instance_attributes>
 </primitive>

 <primitive class="ocf" id="public-net" provider="myResources" \
        type="IPaddr" is_managed="true">
  <instance_attributes id="pub_attributes">
   <attributes>
    <nvpair id="pub_ip" name="ip" value="123.123.123.3"/>
    <nvpair id="pub_mask" name="netmask" value="24"/>
    <nvpair id="pub_nic" name="nic" value="eth0"/>
    <nvpair id="pub_gw" name="gateway" value="123.123.123.254"/>
   </attributes>
  </instance_attributes>
 </primitive>

</group>
</resources>
```

## B.7 Anpassung der Firewall für LVS/NAT und LVS/DR

```
@@ -27,13 +27,14 @@
 }

 xen_services() {
-        $IPTABLES -A FORWARD -i $IFACE_BR -o $IFACE_BR -m physdev \
-               --physdev-in $IFACE_VIF --physdev-out $IFACE_EXT \
-               -s $Trusted_Net -d $UNIVERSE -j ACCEPT
+        $IPTABLES -A FORWARD -i $IFACE_BR -o $IFACE_BR -m physdev \
+               --physdev-in $IFACE_VIF --physdev-out $IFACE_EXT \
+               -s $UNIVERSE -d $UNIVERSE -j ACCEPT
         $IPTABLES -A FORWARD -i $IFACE_BR -o $IFACE_BR -m physdev \
                --physdev-in $IFACE_EXT --physdev-out $IFACE_VIF \
                -d $Trusted_Net -s $UNIVERSE -j ACCEPT
         $IPTABLES -A FORWARD -i $IFACE_BRINT -o $IFACE_BRINT \
                -m physdev --physdev-in $IFACE_VIF --physdev-out \
                $IFACE_VIF -s $Int_Net -d $Int_Net -m state \
                --state NEW,ESTABLISHED,RELATED -j ACCEPT
         $IPTABLES -A FORWARD -i $IFACE_BRINT -o $IFACE_BR \
                -m physdev --physdev-in $IFACE_VIF --physdev-out \
                $IFACE_EXT -s $Int_Net -d $UNIVERSE -m state \
                --state NEW,ESTABLISHED,RELATED -j ACCEPT
         $IPTABLES -t nat -A POSTROUTING -o $IFACE_BR -m physdev \
```

```
                --physdev-out $IFACE_EXT -s $Int_Net -j SNAT \
                --to-source $EXT_IP
        $IPTABLES -A FORWARD -i $IFACE_BR -o $IFACE_BRINT \
                -m physdev --physdev-in $IFACE_EXT --physdev-out \
                $IFACE_VIF -s $UNIVERSE -d $Int_Net -m state \
                --state NEW,ESTABLISHED,RELATED -j ACCEPT
        $IPTABLES -A FORWARD -i $IFACE_BR -o $IFACE_BR \
                -m physdev --physdev-in $IFACE_VIF --physdev-out \
                $IFACE_VIF -s $Trusted_Net -d $Trusted_Net \
                -j ACCEPT
+       $IPTABLES -A FORWARD -i $IFACE_BR -o $IFACE_BR \
               -m physdev --physdev-in $IFACE_VIF --physdev-out \
               $IFACE_VIF -s $Trusted_Net -d 224.0.0.18 -j ACCEPT
 }

 server_net() {
@@ -44,9 +45,9 @@
        $IPTABLES -A INPUT -i $SRV_BR -s $Bond_Net -d $Bond_Net \
                -m physdev --physdev-in $IFACE_BOND -m state \
                --state NEW,ESTABLISHED,RELATED -j ACCEPT
        $IPTABLES -A INPUT -i $SRV_BR -s $Bond_Net -d $Bond_Net \
                -m physdev --physdev-in $IFACE_VIF -m state \
                --state NEW,ESTABLISHED,RELATED -j ACCEPT
        $IPTABLES -A OUTPUT -o $SRV_BR -s $Bond_Net -d $Bond_Net \
                -m physdev --physdev-out $IFACE_VIF -m state \
                --state NEW,ESTABLISHED,RELATED -j ACCEPT
-       $IPTABLES -A FORWARD -i $SRV_BR -o $SRV_BR -s $Bond_Net \
               -d $Bond_Net -m physdev --physdev-in $IFACE_VIF \
               --physdev-out $IFACE_BOND -m state \
               --state NEW,ESTABLISHED,RELATED -j ACCEPT
-       $IPTABLES -A FORWARD -i $SRV_BR -o $SRV_BR -s $Bond_Net \
               -d $Bond_Net -m physdev --physdev-out $IFACE_VIF \
               --physdev-in $IFACE_BOND -m state \
               --state NEW,ESTABLISHED,RELATED -j ACCEPT
-       $IPTABLES -A FORWARD -i $SRV_BR -o $SRV_BR -s $Bond_Net \
               -d $Bond_Net -m physdev --physdev-in $IFACE_VIF \
               --physdev-out $IFACE_VIF -m state \
               --state NEW,ESTABLISHED,RELATED -j ACCEPT
+       $IPTABLES -A FORWARD -i $SRV_BR -o $SRV_BR -s $UNIVERSE \
               -d $UNIVERSE -m physdev --physdev-in $IFACE_VIF \
               --physdev-out $IFACE_BOND -m state \
               --state NEW,ESTABLISHED,RELATED -j ACCEPT
+       $IPTABLES -A FORWARD -i $SRV_BR -o $SRV_BR -s $UNIVERSE \
               -d $UNIVERSE -m physdev --physdev-out $IFACE_VIF \
               --physdev-in $IFACE_BOND -m state \
               --state NEW,ESTABLISHED,RELATED -j ACCEPT
```

```
+        $IPTABLES -A FORWARD -i $SRV_BR -o $SRV_BR -s $UNIVERSE \
                -d $UNIVERSE -m physdev --physdev-in $IFACE_VIF \
                --physdev-out $IFACE_VIF -m state \
                --state NEW,ESTABLISHED,RELATED -j ACCEPT
 }

 log-rules() {
```

## B.8 Firewall-Skript für einen LVS/NAT-Director

Ein Gentoo-Initskript, das auf dem *Director* das notwendige NATting konfiguriert:

```
#!/sbin/runscript

opts="start stop status"

depend() {
        need net
}

start() {
        ebegin "Starting Firewall..."

        echo 1 > /proc/sys/net/ipv4/ip_forward
        for f in /proc/sys/net/ipv4/conf/*/rp_filter
        do
                echo 0 > $f
        done

        $IPTABLES -A FORWARD -i eth1 -s 192.168.0.0/24 -j ACCEPT
        $IPTABLES -t nat -A POSTROUTING -o eth0 -s 192.168.0.0/24 \
                -j SNAT --to-source 123.123.123.1

        eend ${?}
}

stop() {
        ebegin "Everything allowed..."
        $IPTABLES -P INPUT ACCEPT
        $IPTABLES -P OUTPUT ACCEPT
        $IPTABLES -P FORWARD ACCEPT

        $IPTABLES -F
        $IPTABLES -t nat -F
        $IPTABLES -X
```

```
        eend ${?}
}

status() {
        ebegin "Status Information"
        date
        $IPTABLES -v -L
        $IPTABLES -v  -t nat -L
        eend ${?}
}
```

## B.9 Netzwerkkonfiguration der Real Server für LVS/DR

Die Datei /etc/conf.d/net:

```
modules=( "iproute2" )

config_eth0=(
        "123.123.123.10 netmask 255.255.255.0 broadcast \
                123.123.123.255"
)
routes_eth0=(
        "default via 123.123.123.254"
)

config_dummy0=(
       "123.123.123.1 netmask 255.255.255.255 broadcast \
                       123.123.123.1"
)

preup() {
        if [[ "${IFACE}" == "dummy0" || "${IFACE}" == tunl0 ]]
        then
                /sbin/sysctl -w net.ipv4.conf.all.rp_filter="0"
                /sbin/sysctl -w net.ipv4.conf.all.arp_ignore="1"
                /sbin/sysctl -w net.ipv4.conf.all.arp_announce="2"
        fi
}
```

## B.10 Netzwerkkonfiguration der Real Server für LVS/TUN

Die Datei `/etc/conf.d/net`:

```
modules=( "iproute2" )

config_eth0=(
        "123.123.123.10 netmask 255.255.255.0 broadcast \
                123.123.123.255"
)
routes_eth0=(
        "default via 123.123.123.254"
)

config_dummy0=(
       "123.123.123.1 netmask 255.255.255.255 broadcast \
                       123.123.123.1"
)

config_dummy0=(
       "131.159.46.156 netmask 255.255.255.255 broadcast \
                       131.159.46.156"
)

preup() {
        if [[ "${IFACE}" == "dummy0" || "${IFACE}" == tunl0 ]]
        then
                /sbin/sysctl -w net.ipv4.conf.all.rp_filter="0"
                /sbin/sysctl -w net.ipv4.conf.all.arp_ignore="1"
                /sbin/sysctl -w net.ipv4.conf.all.arp_announce="2"
        fi
}
```

## B.11 Konfiguration eines Keepalived-Clusters

Für einen *LVS-Director* (`director1`), der als Loadbalancer für einen Mailserver mit SMTP, IMAPS und POP3S dient, und der gleichzeitig als *Backup-Director* für einen *LVS-Director* fungiert, der die Lastverteilung für einen Webserver übernimmt, könnte eine Keepalived-Konfiguration folgendermaßen aussehen.

```
global_defs {
        notification_email {
                admin@your.domain.name
```

```
        }
        notification_email_from director1@your.domain.name
        smtp_server 123.123.123.1
        smtp_connect_timeout 30
        router_id director1
}

vrrp_sync_group mailserver {
        group {
                mail_vip
                mail_gw
        }
}

vrrp_instance mail_vip {
        state MASTER
        interface eth0
        virtual_router_id 10
        priority 150
        advert_int 1
        authentication {
                auth_type PASS
                auth_pass topSecret
        }
        virtual_ipaddress {
                123.123.123.1
        }
}

vrrp_instance mail_gw {
        state MASTER
        interface eth1
        virtual_router_id 11
        priority 150
        advert_int 1
        authentication {
                auth_type PASS
                auth_pass verySecret
        }
        virtual_ipaddress {
                192.168.0.1
        }
}

vrrp_sync_group webserver {
        group {
```

```
                web_vip
                web_gw
        }
}

vrrp_instance web_vip {
        state BACKUP
        interface eth0
        virtual_router_id 20
        priority 100
        advert_int 1
        authentication {
                auth_type PASS
                auth_pass veryverySecret
        }
        virtual_ipaddress {
                123.123.123.2
        }
}

vrrp_instance web_gw {
        state BACKUP
        interface eth1
        virtual_router_id 21
        priority 100
        advert_int 1
        authentication {
                auth_type PASS
                auth_pass totallySecret
        }
        virtual_ipaddress {
                192.168.0.2
        }
}

virtual_server 123.123.123.1 25 {
        delay_loop 6
        lb_algo rr
        lb_kind NAT
        persistence_timeout 50
        protocol TCP

        real_server 192.168.0.101 25 {
                weight 1
                SMTP_CHECK {
                        connect_timeout 3
```

```
                }
        }

        real_server 192.168.0.102 25 {
                weight 1
                SMTP_CHECK {
                        connect_timeout 3
                }
        }
}

virtual_server 123.123.123.1 993 {
        delay_loop 6
        lb_algo rr
        lb_kind NAT
        persistence_timeout 50
        protocol TCP

        real_server 192.168.0.101 993 {
                weight 1
                TCP_CHECK {
                        connect_timeout 3
                }
        }

        real_server 192.168.0.102 993 {
                weight 1
                TCP_CHECK {
                        connect_timeout 3
                }
        }
}

virtual_server 123.123.123.1 995 {
        delay_loop 6
        lb_algo rr
        lb_kind NAT
        persistence_timeout 50
        protocol TCP

        real_server 192.168.0.101 995 {
                weight 1
                TCP_CHECK {
                        connect_timeout 3
                }
        }
```

```
        real_server 192.168.0.102 995 {
                weight 1
                TCP_CHECK {
                        connect_timeout 3
                }
        }
}

virtual_server 123.123.123.2 80 {
        delay_loop 6
        lb_algo rr
        lb_kind NAT
        persistence_timeout 50
        protocol TCP

        real_server 192.168.0.201 80 {
                weight 1
                HTTP_GET {
                        url {
                                path /index.html
                        }
                        connect_timeout 3
                }
        }

        real_server 192.168.0.202 80 {
                weight 1
                HTTP_GET {
                        url {
                                path /index.html
                        }
                        connect_timeout 3
                }

        }
}
```

Auf dem zweiten *Director* (`director2`) befindet sich eine symmetrische Konfigurationsdatei, bei der im wesentlichen nur alle `router_id`, `state` und `priority` Optionen angepasst sind. Der `state` jedes `vrrp_instance`-Blocks muss von `MASTER` in `BACKUP` oder umgekehrt invertiert werden.

Alle Unterschiede der Datei `/etc/keepalived/keepalived.conf` auf dem zweiten *Director* sind:

```
@@ -2,10 +2,10 @@
        notification_email {
                schroepf@in.tum.de
        }
-       notification_email_from director1@your.domain.name
+       notification_email_from director2@your.domain.name
        smtp_server 123.123.123.1
        smtp_connect_timeout 30
-       router_id director1
+       router_id director2
 }

 vrrp_sync_group mailserver {
@@ -16,10 +16,10 @@
 }

 vrrp_instance mail_vip {
-       state MASTER
+       state BACKUP
        interface eth0
        virtual_router_id 10
-       priority 150
+       priority 100
        advert_int 1
        authentication {
                auth_type PASS
@@ -31,10 +31,10 @@
 }

 vrrp_instance mail_gw {
-       state MASTER
+       state BACKUP
        interface eth1
        virtual_router_id 11
-       priority 150
+       priority 100
        advert_int 1
        authentication {
                auth_type PASS
@@ -53,10 +53,10 @@
 }

 vrrp_instance web_vip {
-       state BACKUP
+       state MASTER
```

```
        interface eth0
        virtual_router_id 20
-       priority 100
+       priority 150
        advert_int 1
        authentication {
                auth_type PASS
@@ -68,10 +68,10 @@
 }

 vrrp_instance web_gw {
-       state BACKUP
+       state MASTER
        interface eth1
        virtual_router_id 21
-       priority 100
+       priority 150
        advert_int 1
        authentication {
                auth_type PASS
```

Um statt LVS/NAT *IP-Tunneling* oder *Direct Routing* zu verwenden, genügt es, alle `lb_kind` Einträge der Datei abzuändern in

```
lb_kind TUN
```

für *IP-Tunneling* über LVS/TUN, bzw. in

```
lb_kind DR
```

für *Direct Routing* über LVS/DR.

Zusätzlich lässt sich die Konfiguration von Keepalive noch vereinfachen, da kein NAT-Gateway mehr benötigt wird. Die `vrrp_instance` Blöcke für die beiden Gateways (also `mail_gw` und `web_gw`) sowie die `vrrp_sync_group`-Blöcke sind für LVS/TUN und LVS/DR nicht notwendig, und können aus der Konfiguration entfernt werden.

## B.12 Heartbeat Ressource für ClusterIP

```
<resources>
 <clone id="clusterip_clone">
  <instance_attributes id="clone_attributes">
   <attributes>
    <nvpair id="clmax" name="clone_max" value="2"/>
    <nvpair id="clnomax" name="clone_node_max" value="2"/>
```

```
   </attributes>
  </instance_attributes>
  <primitive class="ocf" id="cluster_ip" provider="myAgents"
          type="IPaddr2">
   <instance_attributes id="ip_attributes">
    <attributes>
     <nvpair id="clusterip_address" name="ip"
             value="123.123.123.1"/>
     <nvpair id="clusterip_mac" name="mac"
             value="01:02:03:04:05:06"/>
     <nvpair id="clusterip_hash" name="clusterip_hash"
             value="sourceip-sourceport"/>
     <nvpair id="clusterip_nic" name="nic" value="eth0"/>
     <nvpair id="clusterip_netmask" name="cidr_netmask" value="24"/>
    </attributes>
   </instance_attributes>
  </primitive>
 </clone>
</resources>
```

# Literaturverzeichnis


[1] *Annotated CIB DTD (1.0)*. URL, http://www.linux-ha.org/ClusterResourceManager/DTD1.0/Annotated.

[2] *Cisco EtherChannel Tehnology*. URL, http://www.cisco.com/warp/public/cc/techno/lnty/etty/fsetch/tech/fetec_wp.pdf, Februar 2003.

[3] *Crossroads*. URL, http://crossroads.e-tunity.com/.

[4] *DFS Technical Reference*. URL, http://technet2.microsoft.com/WindowsServer/en/library/20ffb860-f802-455c-9ca2-5194f79a9eb41033.mspx, März 2003.

[5] *FRS Technical Reference*. URL, http://technet2.microsoft.com/WindowsServer/en/library/965a9e1a-8223-4d3e-8e5d-39aeb70ec5d91033.mspx, März 2003.

[6] *Ganeti Projekt*. URL, http://code.google.com/p/ganeti/.

[7] *General Parallel File System*. URL, http://www-03.ibm.com/systems/clusters/software/gpfs.pdf, Juli 2006.

[8] *ProLiant networking*. URL, http://h18004.www1.hp.com/products/servers/networking/teaming.html.

[9] *Linux Channel Bonding*. URL, http://sourceforge.net/projects/bonding.

[10] *Linux Virtual Server*. URL, http://www.linuxvirtualserver.org.

[11] *Load-Sharing with iptables in Linux HA*. URL, http://www.multinet.de/LLS/Load-share.pdf.

[12] *LSB Resource Agent*. URL, http://www.linux-ha.org/LSBResourceAgent.

[13] *MySQL Cluster*. URL, http://dev.mysql.com/doc/refman/5.0/en/mysql-cluster.html.

[14] *MySQL Replication*. URL, http://dev.mysql.com/doc/refman/5.0/en/replication.html.

[15] *Network Load Balancing Architecture*. URL, http://technet2.microsoft.com/WindowsServer/en/library/3215a764-bbfb-42bb-aa46-748f27ce2d411033.mspx, Januar 2003.

[16] Schneider, T.: *Virtualisierte UNIX-/Linux-Hosting-Plattform*. Systementwicklungsprojekt, Technische Universität München, Fakultät für Informatik, Garching, Deutschland, November 2006.

[17] *Server Cluster Architecture*. URL, http://technet2.microsoft.com/WindowsServer/en/library/b6d5085d-dd23-44b9-bcec-359f0084ffe81033.mspx, Januar 2003.

[18] *The High Availability Linux Project*. URL, http://www.linux-ha.org.

[19] *Veritas Volume Replicator Option by Symantec*. URL, http://www.symantec.com/enterprise/products/overview.jsp?pcid=1019&pvid=20_1, 2006.

[20] *Virtual Router Redundancy Protocol*. URL, http://www.ietf.org/rfc/rfc2338.txt.

[21] *VMware DRS*. URL, http://www.vmware.com/de/products/vi/vc/drs.html.

[22] *VMware HA*. URL, http://www.vmware.com/de/products/vi/vc/ha.html.

[23] *VMware VMotion*. URL, http://www.vmware.com/de/products/vi/vc/vmotion.html.

[24] *Wikipedia: Computercluster*. URL, http://de.wikipedia.org/wiki/Computercluster.

[25] *Wikipedia: Failover*. URL, http://de.wikipedia.org/wiki/Failover.

[26] *Wikipedia: Hochverfügbarkeit*. URL, http://de.wikipedia.org/wiki/Hochverf%C3%BCgbarkeit.

[27] *Wikipedia: Shared Storage*. URL, http://de.wikipedia.org/wiki/Shared_Storage.

[28] *Xen Virtual Machine Monitor*. URL, http://www.cl.cam.ac.uk/research/srg/netos/xen.