Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich
Informatik

Bachelor-Thesis

im Studiengang Informatik

Automatisierter Failover dynamischer Netzwerkdienste

von

Sebastian Friedrichs

Erstprüfer:	Herr Prof. Dr. Karl Jonas
Zweitprüfer:	Frau Prof. Dr. Kerstin Uhde
Eingereicht am:	12.01.2016

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Formelverzeichnis

Quelltextverzeichnis

Abkürzungsverzeichnis

AEC	Availability Environment Classifications
BSI	Bundesamt für Sicherheit in der Informationstechnik
CARP	Common Address Redundancy Protocol
Carrier Grade	Eine Verfügbarkeit von 99,999%
CIB	Cluster Information Base
Cloud	Cloud Computing
CRM	Cluster Resource Manager
DRBD	Distributed Replicated Block Device
HA	High Availability
IEEE	Institute of Electrical and Electronics Engineers
ISP	Internet Service Provider
LSB	Linux Standard Base
LVS	Linux Virtual Server
MTBF	Mean Time Between Failure
MTTF	Mean Time To Failure
MTTR	Mean Time To Repair
OCF	Open Cluster Framework
OS	Operation System
OSI	Open Systems Interconnection Model
OSPF	Open Shortest Path First
RAM	Random-Access Memory; Arbeitsspeicher
RHEL	Red Hat Enterprise Linux
SIP	Session Initiation Protocol
SLA	Service Level Agreement(s)
SPoF	Single Point(s) of Failure
SS7	Signaling System No. 7
STP	Signaling Transfer Point(s)
STONITH	Shoot The Other Node In The Head
Tier	Hierarchieebene
VM	Virtuelle Maschine(n)
VRRP	Virtual Router Redundancy Protocol

1 Einleitung

Im Jahre 2015 ist es mehr und mehr zu einer Selbstverständlichkeit geworden, dass Dienste und Funktionen überall und jederzeit verfügbar sind. Dabei existiert diese Erwartungshaltung beim Endverbraucher wie in Unternehmen. Der Trend hin zum Cloud Computing (Cloud) bildet damit eine Synergie. Das Bundesamt für Sicherheit in der Informationstechnik (BSI) definiert Cloud Computing als ein „Modell, das es erlaubt bei Bedarf, jederzeit und überall bequem über ein Netz auf einen geteilten Pool von konfigurierbaren Rechnerressourcen (z. B. Netze, Server, Speichersysteme, Anwendungen und Dienste) zuzugreifen, die schnell und mit minimalem Managementaufwand oder geringer Serviceprovider-Interaktion zur Verfügung gestellt werden können.“ [Inf]. Beachtenswert in Bezug auf diese Arbeit ist das Wort *jederzeit*, welches im späteren Verlauf genauer spezifiziert werden wird. Um diesem Anspruch zu genügen, sind Sicherungsmaßnahmen seitens der Anbieter erforderlich. Ziel der Arbeit ist daher die Konzeption und Umsetzung einer automatisierten Failover-Lösung, die möglichst viele Informationen und Zustände der überwachten Applikationen erhält.

Diese Anforderung an Dienste beschäftigt auch die Parteien, welche für die Festlegung der Thematik dieser Thesis ausschlaggebend waren: Die finocom AG, ein Anbieter von Telekommunikationsanlagen aus der Cloud mit Sitz in Köln, und das Media-Communication Lab der Hochschule Bonn-Rhein-Sieg unter Leitung von Herrn Professor Doktor Karl Jonas. In der Telekommunikationsbranche spielt die permanente Verfügbarkeit eine hervorgehobene Rolle. Dies führt logischerweise dazu, dass besonders im IT-Sektor entsprechende Maßnahmen ergriffen werden müssen, um Verfügbarkeit garantieren zu können. Bei Cloud-Applikation sowie -Services handelt es sich vorrangig um sogenannte Netzwerkdienste. Also um jene Dienste, die über ein Netzwerk, möglicherweise sogar das Internet, erreichbar sind. Einige sind statischer Natur und speichern nur Informationen oder stellen diese zur Verfügung, andere verarbeiten Echtzeitdaten und können viele verschiedene Zustände annehmen. Während es durchaus ausgereifte Methoden zur Herstellung von Redundanz auf Dienst- oder Systemebene für statische Informationen gibt, ist die Erhaltung dynamischer Daten ein relevantes Thema in Forschung und Technik im Jahr 2015. Verschiedene Forscher und Anbieter von Lösungen für Hochverfügbarkeit versuchen sich dem Ziel des automatisierten, vollständig stateful Failover von Diensten zu nähern. Dabei steht stateful für die Übernahme möglichst vieler Daten und Zustände und das namensgebende Failover für die Übernahme von Aufgaben einer Instanz A durch eine zweite Instanz B im Fehlerfall. In der Forschung liegt der Fokus meist nicht mehr auf dem Failover

eines Nodes selbst, sondern auf der Unabhängigkeit eines Dienstes von der Verfügbarkeit eines einzelnen Nodes. Ein Node steht für ein Gerät, beziehungsweise ein System, welches das Internet Protokoll implementiert. In vorliegendem Fall treten Nodes in Form eines physikalischen Servers oder einer virtuellen Maschine (VM) und analog zu Host auf. Ein Host wiederum ist ein Gerät, welches das Internet Protokoll implementiert und einen oder mehrere Dienste anbietet [IET98, *nach S.3*]. Die Forschung befindet sich hier auf der nächst höheren Abstraktionsebene. Einige Beispiele dazu sind die Whitepaper „Strategies to Solve and Optimize Management of Multi-tiered Business Services“ von Symantec [Sym13], „Aura Application Enablement (AE) Services“ von Avaya [Ava14] oder das auf IPv6 setzende „Fast, Secure Failover for IP“ der University of St Andrews [Pho14]. Diese wurden innerhalb der letzten drei Jahre verfasst.

Die Motivation für die vorliegende praxisorientierte Thesis besteht darin, dass digitale Dienstleister ihre Services und Systeme ausfallsicher wie redundant auslegen müssen. Nur so können garantierte Verfügbarkeiten, die bei Vertragsabschluss in sogenannten Service Level Agreements (SLA) festgehalten werden, dauerhaft erreicht werden. Es existieren grundsätzlich zwei verschiedene Lösungsansätze für diese Problematik, die freien Open Source und die kommerziellen Modelle. Ein Teil-Ziel der Arbeit soll ein Überblick über die aktuelle Situation am Markt und in der Technik sein. Darauf folgend werden die Ansätze gegeneinander abgewogen, ausgewählt und zu einer Gesamtlösung kombiniert. Dabei sind spezifische Anforderungen ebenso zu beachten, wie die Erstellung von Auswertungsschemata für eine spätere Bewertung.

Es folgt die Herleitung einer konkreten Fragestellung und die Abgrenzung gegen ihr unmittelbares Umfeld. Als erstes Problem ist der Ausfall eines Nodes zu betrachten. Der Fokus liegt auf der möglichst zeitnahen, erneuten Verfügbarkeit eines von diesem Node angebotenen Dienstes, wobei keine Schritte seitens des Dienstnehmers notwendig sein dürfen. Für einen solchen Dienst soll eine minimale Ausfallzeit erreicht werden, unabhängig von der Zeitspanne, die für eine Reparatur des Nodes benötigt wird. Daraus folgt, dass eine Netzarchitektur mit mehreren, unabhängigen Nodes hergeleitet werden muss. Ein Node B übernimmt automatisch die Aufgaben, welche bis zu seinem Ausfall Node A inne hatte. Hier wird der Fokus auf den Erhalt möglichst vieler Zustände des Netzwerkdienstes gelegt und darauf, auf welche Art und Weise dies bewerkstelligt werden kann. Zusätzlich wird der jeweilige Aufwand in die Einteilung der Ziele in obligatorische und optionale Erfolgsbedingungen einfließen. Für die praktische Realisierung des erarbeiteten Konzepts sind der SIP Proxy Kamailio und die Heimautomatisierungssteuerung openHAB vorgegeben. Ersterer wird in der produktiven Umgebung des beteiligten Unternehmens eingesetzt und openHAB bietet aufgrund weiterer Forschungsarbeiten eine Relevanz für die Hochschule Bonn-Rhein-Sieg. SIP ist die Kurzform des Session Initiation Protocol (SIP), das vornehmlich in der VoIP-Telefonie eingesetzt wird und Proxy steht für einen Vermittler zwischen zwei Endstellen oder Systemen.

Grundlegende Teilaspekte sind also die Folgenden:

- Ein Node A fällt während des Betriebs aus
- Der Ausfall wird nicht bemerkt
- Ein Netzwerkdienst ist nach dem Ausfall nicht mehr verfügbar
- Der Dienst wird nicht automatisch wiederhergestellt
- Welche Single Points of Failure sind zu beachten

Der Fokus dieser Arbeit liegt auf diesen weiterführenden Problematiken:

- Nach einem Failover sind Daten und Zustände eines Dienstes nicht verfügbar
- Anwendungsfall 1: Kamailio SIP Proxy ist nicht verfügbar
- Anwendungsfall 1: Node B hat keinen Zugriff auf die von Node A erhobenen Zustände
- Anwendungsfall 2: openHAB ist nicht verfügbar
- Anwendungsfall 2: Instanz B hat keinen Zugriff auf die von Instanz A erhobenen Daten

Die Zielsetzung der Arbeit wird durch ein Failover-Cluster mit mindestens zwei Nodes abgebildet, das möglichst keinen neuen Single Point of Failure (SPoF) generieren soll und wobei Cluster für einen Verbund von 2 bis n Nodes steht. SPoF ist der Fachterminus für Verbindungsknoten, die bei einem Ausfall eine Kommunikation und eventuelle Funktionalität anderer Komponenten unmöglich machen. Dabei ist diese Grundstruktur von den anderen Teilen des Netzwerks abzugrenzen, wobei jene jedoch benannt werden sollen. Dieses Vorgehen ist zwingend notwendig, da ein Netzwerk aus diversen Komponenten besteht. Ein tieferes Verständnis und die Möglichkeiten ihres redundanten Aufbaus bieten Material und Problemstellungen für jeweils eigene Abhandlungen. Auch treffen sie nicht den Kern der Fragestellung dieser Arbeit und nutzen teils proprietäre Betriebssysteme und Firmware. Auf dem Cluster aufbauend wird die Verfügbarkeit der angebotenen Netzwerkdienste maximiert. Es sollen möglichst viele Informationen und Status erhalten bleiben. Aus der Auswertung des Praxisteils geht hervor, inwiefern eine gewählte Lösung die gesetzten Ziele erreicht hat und auf welche Art und Weise die Bearbeitung der Thematik weitergeführt werden könnte.

Die Struktur der Arbeit wurde wie oben genannt gewählt, um eine logische Abfolge von Motivation, Begriffsklärung, Marktsituation, Überblick zum Stand der Technik, Auswahl einer Lösung sowie deren Anwendung in der Praxis mit anschließender Auswertung zu ermöglichen. Ein persönliches Fazit der Thematik inklusive Ausblick schließt die Thesis ab. Als primäre Quellen wurden die dritte Auflage des Buches „Clusterbau: Hochverfügbarkeit mit Linux“ von Michael Schwartzkopff [Sch12] aus dem Jahre 2012 sowie „Linux Hochverfügbarkeit: Einsatzszenarien und Praxislösungen“ von Oliver Liebel [Lie13] in zweiter Auflage gewählt. Hinzu kommen diverse Onlinequellen, Definitionen des Institute of Electrical and Electronics Engineers (IEEE) und Zahlen zur Marktsituation im Jahr 2015.

2 Stand der Technik

2.1 Begriffsklärung

An dieser Stelle sollen thematisch relevante Begriffe und Teilgebiete erläutert werden, um Eindeutigkeit bei deren späterer Verwendung gewährleisten zu können. Zunächst werden die im Titel enthaltenen Termini ausgeführt, danach auf weitergehende eingegangen.

2.1.1 Verfügbarkeit

In der Einleitung wird bereits mehrfach von Verfügbarkeit gesprochen. Der Duden definiert Verfügbarkeit als den „Grad, in dem etwas verfügbar ist“ [Dud, *Verfügbarkeit*]. Das Adjektiv verfügbar wird wiederum mit „für den sofortigen Gebrauch [...] vorhanden“ beschrieben [Dud, *verfügbar*]. Dies trifft auch auf die Bedeutung des Begriffes innerhalb der IT-Branche zu. Hier bezieht sich jene im Besonderen auf die Möglichkeit des sofortigen Zugriffs auf Dienste, Dienstleistungen, Software oder Hardware. *Sofort* steht dabei je nach vertraglicher Regelung für eine am Empfinden des Nutzers gemessene kurze bis nicht wahrnehmbare Zeitspanne oder einen schriftlich festgehaltenen Wert in der Einheit Zeit.

Dieser schriftlich festgehaltene Wert ist von unterschiedlichen Stellen weiter unterteilt worden. Die Verfügbarkeitsklassen des IEEE ähneln den in der Einleitung gezeigten Darstellungen. Ein Jahr entspricht genau 360 Tagen.

Verfügbarkeits-Klasse	Verfügbarkeit	Ausfallzeit pro Jahr
2	99%	3,60 Tage
3	99,9%	8,76 Stunden
4	99,99%	52 Minuten
5	99,999%	5 Minuten
6	99,9999%	30 Sekunden
7	99,99999%	3 Sekunden

Tabelle 2.1: IEEE Verfügbarkeitsklassen, nach [Lie13, *S.32*]

Tiefergehende Informationen zu Berechnungen, Kennzahlen und Werten bietet das IEEE unter anderem in seinem Dokument IEEE762TF [IEE06].

Die Harvard Research Group unterteilt Verfügbarkeit dagegen in sechs durch Worte beschriebene Klassen, die Availability Environment Classifications (AEC) [Lie13, *S.32*].

Verfügbarkeits-Klasse (AEC)	**Bezeichnung**	**Beschreibung**
AEC-0	Conventional	Funktion kann nicht unterbrochen werden. Datenintegrität ist nicht essentiell.
AEC-1	Highly Reliable	Funktion kann unterbrochen werden, Datenintegrität muss jedoch gewährleistet sein.
AEC-2	High Availability	Funktion darf nur innerhalb festgelegter Zeiten oder zur Hauptbetriebszeit minimal unterbrochen werden.
AEC-3	Fault Resilient	Funktion muss innerhalb festgelegter Zeiten oder während der Hauptbetriebszeit ununterbrochen aufrechterhalten werden.
AEC-4	Fault Tolerant	Funktion muss ununterbrochen aufrechterhalten werden, 24/7-Betrieb (24 Stunden, 7 Tage die Woche) muss gewährleistet sein.
AEC-5	Disaster Tolerant	Funktion muss unter allen Umständen verfügbar sein.

Tabelle 2.2: Harvard Research Group AECs, nach [Lie13, *S.33*]

Geht man „von den Spezifikationen der Tabellen aus, würde bereits eine durchschnittliche Downtime von 3,6 Tagen pro Jahr in die Klassifikation AEC-2, High Availability [(HA) – Anm. d. Verf.] fallen, entsprechend einer Downtime von gut 1,5 Stunden pro Woche. Fakt ist natürlich, dass jede kleine 9 hinter dem Komma [...] in Euro eine ganze Menge mehr an Zahlen vor dem Komma kostet.“ [Lie13, *S.33*].

Die in Figur 2.1 ersichtliche Auswirkung der Zahlen hinter dem Komma auf die Größe der maximal erlaubten Ausfallzeit soll durch folgende Abbildung weiter konkretisiert werden. Für ein Jahr werden auch hier 360 Tage herangezogen.

Verfügbarkeit	Auszeit pro Monat	Auszeit pro Jahr
99,00%	7,20 h	87,66 h
99,50%	3,60 h	43,83 h
99,90%	0,72 h	8,77 h
99,99%	4,32 min	52,59 min
99,999%	0,43 min	5,26 min

Tabelle 2.3: Clusterbau-Verfügbarkeit, nach [Sch12, *S.2*]

Während knapp neun Stunden noch eine gewisse Reaktionszeit bei einem einzigen Ausfall pro Jahr ermöglichen, stellen fünf Minuten bereits ein Problem für die manuelle Fehlerbehebung dar. Die Auswirkungen mehrerer Ausfälle oder größerer Schäden können durch einen redundanten Aufbau des Dienstes oder eine automatisierte Ausfallsicherung abgefedert werden. Dies ermöglicht eine nahezu durchgehende Vertragserfüllung, während kein akuter Zeitdruck mehr auf der Fehlerbehebung liegt. Im Idealfall bemerkt der Kunde einen Ausfall nicht einmal oder ist zumindest nicht in der Lage diesen zu belegen, um Vertragsstrafen durchsetzen zu können.

Der Fachterminus für diese Ausfallzeit lautet Mean Time To Repair (MTTR), also die durchschnittliche Zeit die benötigt wird, um die Funktionalität eines Dienstes wiederherzustellen [Sch12, *S.2ff*]. Diese macht noch deutlicher, wie essentiell ein automatisierter Failover im Jahre 2015 sein kann. Ein solcher kann die MTTR nämlich auf ein Minimum reduzieren, unabhängig von Ausmaß und Art des Fehlers. Weitere Kennzahlen sind die Mean Time Between Failure (MTBF) und die Mean Time To Failure (MTTF), auf welche hier nicht näher eingegangen wird. Als IT-Dienstleister sollte also sehr gut überlegt sein, in welcher Güte man die Verfügbarkeit seiner Dienste anbietet und wie man diese garantieren kann.

2.1.2 Dienst

Zwischen Personen beschreibt der Begriff *Dienst* die Bearbeitung einer bestimmten Aufgabe, die eine Person A anbietet und welche von beliebig vielen anderen Personen in Anspruch genommen werden kann. Im beruflichen Umfeld wird er in der Regel durch ein Entgelt abgegolten [Dud, *Dienst*]. Aus der Definition lässt sich folgern, dass auch Computersysteme Dienste anbieten können, welche von Personen sowie anderen Systemen in Anspruch genommen werden können. „Ein Provider und sein Kunde werden in erster Linie an die Möglichkeit des Transports von IP-Paketen zwischen Hausanschluss und beliebigen anderen Nodes im Internet denken.“ [Sch12, *S.4*]. Ein Provider ist eine Gesellschaft zum Betrieb von Telekommunikationsnetzen [Eur15, *S.3*]. Die Funktion eines Dienstes soll innerhalb dieser Arbeit durch die Aufgabe definiert sein, die ein solcher auf einem Server erfüllt und durch die Leistung, welche er für Dritte anbietet und erbringt.

2.1.3 Dynamischer Netzwerkdienst

Ein erster Stolperstein zum Verständnis findet sich bereits im Titel der hier behandelten Thematik: Der *dynamische Netzwerkdienst*. Nachdem im vorherigen Abschnitt der Dienst an sich definiert wurde, soll nun auf die verbleibenden Begriffe eingegangen werden. Ein Netzwerkdienst ist ein Dienst, der von einem Node A über ein beliebiges Netzwerk angeboten und durch ein bis n Nodes in Anspruch genommen werden kann. Dabei existieren solche, die sich wenig verändern und stets die gleiche Funktionalität bieten. Und es gibt jene, die eine Vielzahl an Zuständen halten sowie die Art ihrer Leistung an diese anpassen müssen. Beispielsweise muss ein SIP-Proxy verschiedene Phasen einer Anruf-Signalisierung durchlaufen und, je nachdem in welcher er sich befindet, auf bestimmte Befehle reagieren sowie andere abweisen.

Soll ein solcher dynamischer Dienst unter Erhaltung möglichst vieler Zustände durch einen Failover-Prozess geleitet werden, reicht eine gemeinsame Datenbank der beteiligten Instanzen möglicherweise nicht mehr aus. Einen statischen Netzwerkdienst redundant zu strukturieren, kann bereits mit regelmäßigen 1:1 Kopien des aktiven Nodes möglich sein. Interessant wird es jedoch, wenn ein dynamischer Netzwerkdienst zu einem spezifischen Zeitpunkt mit den aktuellen Informationen und Zuständen im Fehlerfall mit möglichst geringen Auswirkungen übergeleitet werden muss.

2.1.4 Failover

Ein Dienst wird von einem Node A angeboten. Nun ist dieser Node nicht mehr verfügbar, der Dienst wird jedoch weiter angeboten. Das ist möglich, indem ein Node B die Aufgabe des ersten Nodes übernimmt und den Dienst weiter anbietet. Dabei ist für den Nutzer nicht relevant, welcher Node seine Anfrage letztendlich bearbeitet [RH03, nach *S.5*]. Damit wird die Verfügbarkeit des Dienstes von der eines einzelnen Nodes entkoppelt. Die in der Einleitung erwähnte reale Ausfallzeit des Dienstes sollte dabei möglichst gering ausfallen. Es bleibt dem Anbieter ein beliebig großes Zeitfenster, um den Schaden an Node A zu reparieren und diesen gegebenenfalls wieder in Betrieb zu nehmen.

2.1.5 Automatisierter Failover

Um die reale Ausfallzeit weiter zu minimieren, kann ein solcher Failover automatisiert geschehen. Das heißt, dass die Nodes untereinander eine Kommunikationsstruktur benötigen und in die Lage versetzt werden müssen, unter festgelegten Bedingungen, den Platz des jeweils anderen einnehmen zu können. Aus der Sicht des OSI-Modells wird dies durch eine zusätzliche Schicht oberhalb der Vermittlungsschicht realisiert. Das Open System Interconnections

Model (OSI) beschreibt die Architektur von Computer-Netzwerken mittels Schichten, denen jeweils bestimmte Aufgaben zugeordnet sind [Uni94]. So existiert beispielsweise eine sogenannte Service-IP-Adresse, welche mittels dieser Struktur dynamisch dem Node zugewiesen wird, der eine Anfrage bearbeiten soll [Lie13, *S.224*]. Eine Service-IP entspricht also einer statischen IP-Adresse, die mittels eines Dienstes dynamisch zugewiesen wird. Für weitere Informationen zu dem OSI-Modell der Netzwerktechnik sei an diesem Punkt auf Abschnitt 3.6.2 verwiesen. Für die Kommunikation der Nodes untereinander gibt es unterschiedliche Ansätze & Szenarien sowie proprietäre und Open Source Protokolle und Tools. Kommerzielle Anbieter setzen dabei meist auf vollständige, von ihnen angebotene Betriebssysteme oder Software-Suites. Sowohl unter den freien als auch den kommerziellen Lösungen dominiert Linux als Basis-Betriebssystem. Dabei sind erstere jedoch modular aufgebaut und bieten verschiedene Möglichkeiten der Zusammenstellung. Beginnend mit den strukturellen Ansätzen werden in den späteren Abschnitten weitere Details recherchiert, Möglichkeiten gegeneinander abgewogen und gegebenenfalls ausgewählt.

2.1.6 Loadbalancing

Das so genannte Loadbalancing erweitert oben genannte Mechanismen um eine sinnvolle Verteilung von Anfragen und Last zwischen aktiven Nodes. Dabei kann für die Steuerung der Last die bereits vorhandene Kommunikationsstruktur der Nodes auf der Cluster-Schicht genutzt werden. „Auf diese Art können eingehende Client-Request von multiplen, gleichartig konfigurierten Maschinen bedient werden. Die Verteilung der eingehenden Requests erfolgt in der Regel über eine redundant ausgelegte, zentrale Instanz.“ [Lie13, *S.221*]. Diese Funktionalität wird von einigen Failover Lösungen mit angeboten, bildet jedoch keine Schnittmenge mit dem eigentlichen Ziel dieser Arbeit. Daher soll auf Loadbalancing an sich hier nicht näher eingegangen werden.

2.1.7 Shared Storage

Shared Storage kann eins zu eins aus dem Englischen übersetzt werden und steht für Festplattenspeicher, den sich beliebig viele Nodes teilen. Dadurch kann Node A Daten schreiben und Node B diese lesen, ohne dass weiterer Aufwand nötig wird. Mit Ausnahme davon, dass ein Management der Lese- und Schreibzugriffe etabliert werden muss. Dabei muss beachtet werden, dass ein Shared Storage einen SPoF darstellt und ebenfalls redundant ausgelegt werden sollte. Red Hat Enterprise Linux benötigt für das Verschieben von Instanzen und ähnlichen Vorgängen zwingend diese Art von Speicher, was die Failover-Lösung von Red Hat für Netzwerk-Architekten mit der Vorstellung von vollständig unabhängigen Nodes unattraktiv macht. Das ist vor allem deshalb relevant, weil deren freie Alternativen auf Linuxbasis, bei-

spielsweise Cent-OS oder Debian, diese Einschränkung nicht vornehmen. Die Quelle für diese Informationen ist der Autor selbst, beziehungsweise der Red Hat Premium Support, mit dem der Verfasser innerhalb einer Testphase deren Produkts Enterprise Linux kommuniziert hat. In Verbindung mit der Thematik dieser Arbeit macht ein Shared Storage vor allem dann Sinn, wenn ein Dienst die zu erhaltenen Daten auf dem Festplattenspeicher ablegt.

2.1.8 Shared Databases

Zuletzt soll auf die Möglichkeit eingegangen werden, eine oder mehrere Datenbanken gemeinsam zu verwenden. Im Gegensatz zum Shared Storage wird hier eine kompatible Applikation vorausgesetzt. Der betriebene Dienst muss eine Möglichkeit anbieten, seine Daten in eine Datenbank auszulagern, anstatt sie ausschließlich im Arbeits- oder Festplattenspeicher zu halten. Wichtig ist jedoch nicht nur die Funktionalität des Daten-Exports, sondern auch die des Imports. Im Kontext eines Failover-Clusters, bei dem ein zweiter Node B auf die von Node A erhobenen Daten zugreifen können soll, macht eine solche Struktur nur Sinn, wenn die relevanten Informationen in eine Datenbank geschrieben und von dort auch wieder aktiv in die Bearbeitungsprozess eingelesen werden können. Für weitere Informationen zu Hochverfügbarkeit bei Datenbanken sei an dieser Stelle auf das Kapitel 2.2.3 verwiesen.

2.1.9 Split-Brain

„Eine sogenannte Split-Brain-Situation eines Clusters ist das Ergebnis einer Störung der Kommunikation zwischen den Knoten [...]." [Sch12, *S.21*]. Knoten steht in diesem Falle für Node oder Host. „In einem typischen Cluster beherbergt jeder Node eine Kopie der Clusterdatenbank [...] und damit des eigentlichen Cluster-Gehirns." Der Fachterminus für Clusterdatenbank lautet „Cluster Information Base" (CIB) [Lie13, *S.226ff*]. Alle beteiligten Nodes gehen davon aus, dass der jeweils andere ausgefallen sei und beginnen damit, dessen Funktionalitäten anzubieten. „Im schlimmsten Fall wären nun schreibende Zugriffe über zwei verschiedene Nodes (oder Teilcluster) auf ein und denselben Datenbestand möglich, obwohl dieser gegebenenfalls [...] nicht dafür ausgelegt ist." [Lie13, *S.226ff*]. Dies kann zu einer vollständigen Zerstörung des Datenbestandes führen [Sch12, *S.21*]. Ein mögliche Prävention dieser Zustände ist das sogenannte Quorum.

2.1.10 Quorum

Zerfällt ein Cluster in einzelne Nodes oder Teil-Cluster, kann ein Quorum-Prozess dafür sorgen, dass die vorher beschriebene Split-Brain-Situation nicht eintritt [Sch12, *S.21*]. „Das Prinzip

ist recht einfach: Alle Nodes führen zunächst eine Zählung der aktiven Cluster-Member durch [...], vergleichen sie und geben - wenn erforderlich - sich selbst als Quorum-fähigen Cluster bekannt. Der Teil-Cluster mit der Mehrheit ($> n/2$) gewinnt [...]“ [Lie13, *S.229ff*]. Das heißt, es gewinnt das Teil-Cluster, welches mehr als die Hälfte der Teilnehmer des ehemaligen, vollständigen Cluster beinhaltet. Bei einem Cluster mit genau zwei Nodes ergibt sich daraus immer eine Pattsituation. Da in diesem Fall keine Mehrheitsentscheidung möglich ist, müssen zusätzlich externe Entscheider eingesetzt werden, beispielsweise die sogenannte STONITH-Technik [Lie13, *S.228*]. In aktuelleren Versionen der Linux-Hochverfügbarkeit ist zusätzlich das Pluggable Quorum Framework implementiert, das oben genannte Entscheidung mittels eines Quorum und weiteren Informationen über das Netzwerk trifft [Sch12, *S.22*].

2.2 Strukturelle Ansätze

Für Failover-Cluster existieren zwei unterschiedliche grundsätzliche Strukturen, die wiederum in Unterkategorien aufteilbar sind. Zunächst wird in Active/Passive und Active/Active Szenarien unterschieden. Da eine jener Strukturen für die praktische Umsetzung ausgewählt werden muss, soll dieser Abschnitt einen Überblick über die vorhandenen Möglichkeiten bieten.

2.2.1 Active/Passive

„Active/Passive-Cluster mit zwei oder mehr Nodes dienen ausschließlich der Hochverfügbarkeit bzw. der Ausfallsicherheit und somit der Redundanz der angebotenen Services.“. Sie werden auch Hot-Failover-Cluster genannt [Lie13, *S.224*]. „In einem typischen 2-Node-Cluster-Active/Passive-Setup ist genau ein Node aktiv, auf dem die entsprechenden Dienste bzw. Ressourcen [...] laufen. Der aktive Node ist dabei für die Clients über eine Service-IP erreichbar, die ebenfalls als Cluster-Ressource implementiert ist. Fällt der aktive Node oder einer seiner Services aus, übernimmt der zweite [...] Node die Ressourcen des ersten, inklusive der Service-IP. Somit bedient immer nur genau ein Node über eine IP-Adresse die Anfragen der Client-Applikationen und User.“ [Lie13, *S.224*]. Dieser Ansatz kann in Cold- und Hot-Standby unterteilt werden.

Cold-Standby

In einem Cold-Standby Szenario existieren ein aktiver Node A und ein sich üblicherweise im Standby- oder Sleep-Modus befindender Node B, der entweder durch die Cluster-Software oder sogar den Administrator selbst gestartet werden muss. In jedem Fall ist der zu betrachtende Dienst auf dem zweiten Node nicht aktiv. Da Dienst und gegebenenfalls Node während eines Failover-Prozesses zunächst gestartet und hochgefahren werden müssen, resultiert das Cold-Standby Verfahren in hohen Transitions-Zeiten und in logischer Konsequenz somit in einer großen Zeitspanne zwischen Beginn und Abschluss eines Failovers [Lie13, *S.224*].

Die folgenden Abbildungen zeigen ein Cold-Standby 2-Node-Active/Passive-Setup vor und nach einem Ausfall.

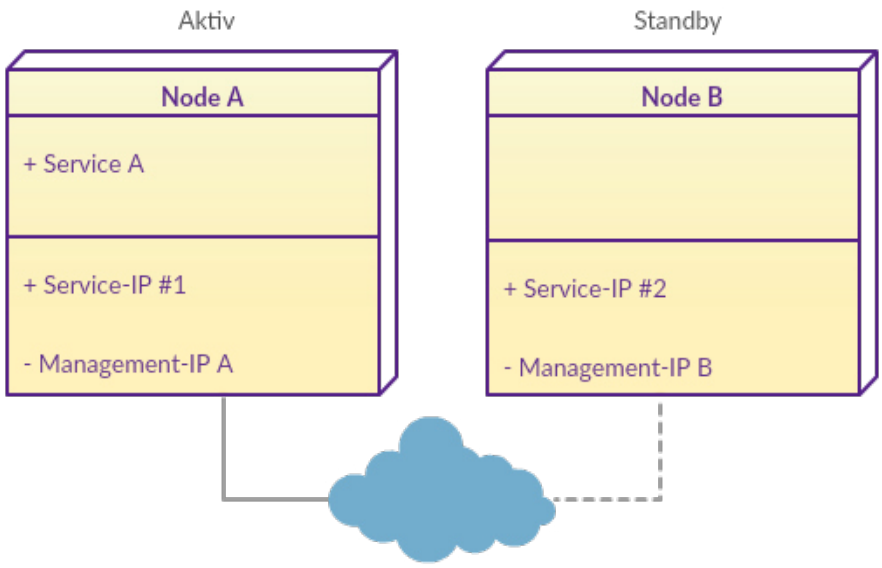

Abbildung 2.1: Cold-Standby pre Failover

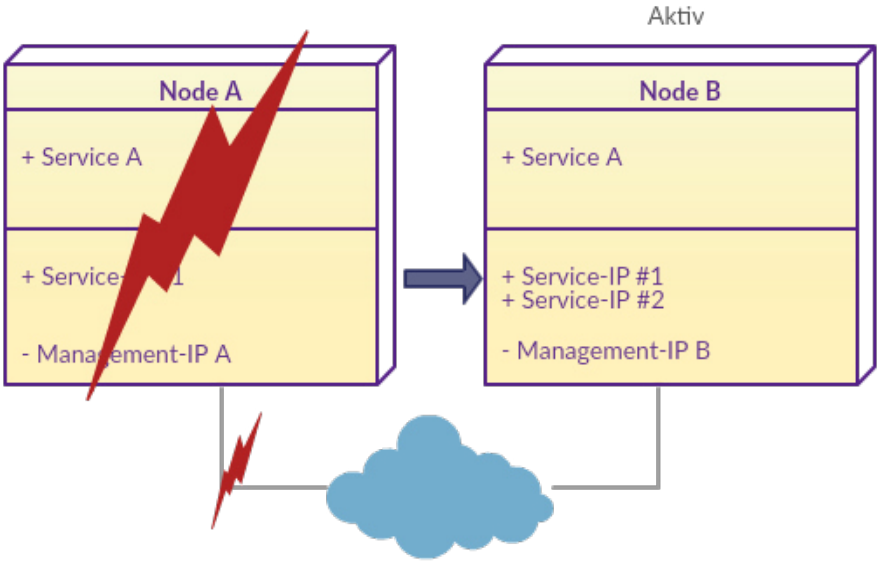

Abbildung 2.2: Cold-Standby post Failover

Hot-Standby

„Active/Passive-Cluster bieten neben der klassischen, aber in der Regel mit hohen Transitions-Zeiten verbundenen Variante der einfachen Service-Übernahme (Fehler auf Node 1 -> Neustart des gleichen Dienstes auf Node 2) die hochverfügbarkeitstechnisch [sic] weitaus bessere des Hot Standy bzw. Hot-Failover - eine der elegantesten, schnellsten und sichersten Lösungen im HA-Bereich. Vereinfacht bedeutet dies: Die Ressource ist mit Hilfe von so genannten Clone-Sets bereits auf allen Nodes des Clusters gestartet und der zweite Node ist nach Übernahme der Service-IP (regulär innerhalb von wenigen Sekundenbruchteilen) sofort mit allen Ressourcen aktiv und für die Clients verfügbar.“ [Lie13, *S.224*]. Das Hot-Standby Prinzip wird auch von dem für Router konzipierten Virtual Router Redundancy Protocol (VRRP) und dessen freiem Pendant Common Address Redundancy Protocol (CARP) angewandt. Dabei werden mehrere physikalische Router oder sogar Nodes zu einer virtuellen Gruppe zusammengefasst und dem Umfeld als ein logischer Netzwerk-Teilnehmer präsentiert. Fällt der aktive aus, übernimmt ein bisher passiver Router die virtuelle IP-Adresse sowie die virtuelle MAC-Adresse, die den Nutzern bekannt sind [Gro, *5798*]. Ein weiterer Vertreter dieser Protokolle ist das Cisco Hot Standby Router Protocol, auch HSRP, der Firma Cisco.

Die folgenden Abbildungen zeigen ein Hot-Standby 2-Node-Active/Passive-Setup vor und nach einem Ausfall.

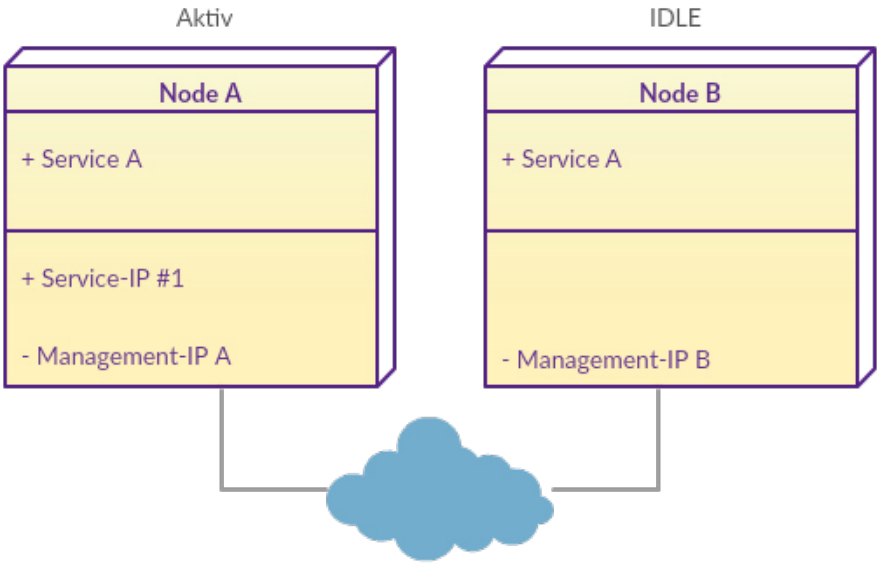

Abbildung 2.3: Hot-Standby pre Failover

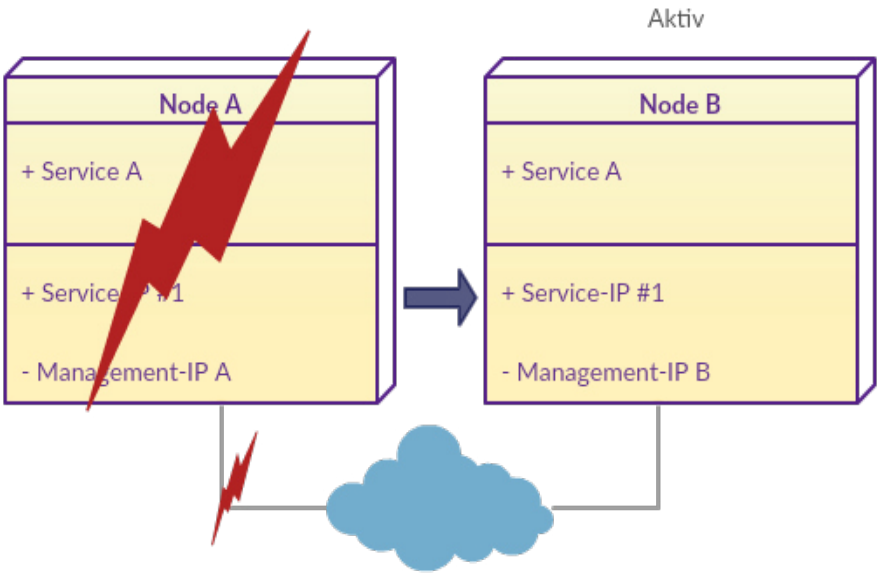

Abbildung 2.4: Hot-Standby post Failover

Vorteile dieses Ansatzes sind eine „hohe Verfügbarkeit und sehr gute Performance. Da beide Maschinen mit exakt den gleichen Services konfiguriert sind, ist die zu erwartende Downtime durch Restart/Übernahme (Hot-Standby) der Services in der Regel deutlich niedriger als eine vollständige Service-Migration innerhalb einer Active/Active-Konfiguration. Zudem kann in einem Active/Passive-Cluster jeweils ein Node zu Wartungs- oder Upgrade-Zwecken tem-

porär deaktiviert werden.“. Es existieren jedoch auch Nachteile: Der passive Node bedeutet Anschaffungs- sowie Wartungskosten und wird dabei im Idealfall niemals einen praktischen Nutzen für den Betreiber bringen, außer dass er die statistische Verfügbarkeit erhöht und die Wartbarkeit vereinfacht [Lie13, *S.224*]. Aus betriebswirtschaftlicher Sicht könnte dies jedoch, zumindest anteilig, von dem höheren möglichen Maximalpreis des angebotenen Dienstes amortisiert werden. Dazu kommt der Umstand, dass man nie 100-prozentig sicher sein kann, dass ein passiver Node im Failover-Fall korrekt funktioniert. Da dieser Tage, Monate und vielleicht Jahre in seinem Standby-Zustand verbleibt oder der laufende Dienst nicht genutzt wird, sind regelmäßige Testläufe notwendig, um die Integrität des passiven Nodes gewährleisten zu können. Auch muss sichergestellt sein, dass die eigene, aktuelle Rolle, also aktiv oder passiv, allen beteiligten Nodes bekannt ist und von diesen wie erwartet umgesetzt wird. Siehe dazu auch die Abschnitte 2.1.9 und 2.1.10.

2.2.2 Active/Active

„In einem Active/Active-Cluster sind beide bzw. alle Nodes des Clusters aktiv und stellen gleiche oder unterschiedliche Ressourcen zur Verfügung, letztere können dabei jedoch im Ausfall- bzw. Failover-Fall [...] üblicherweise von einem anderen Node übernommen werden. Eine Active/Active-Konfiguration könnte z.B. eine Datenbank bedienen, die in mehreren Instanzen auf allen Nodes aktiv ist. Fällt einer der Nodes aus, übernimmt ein anderer Node die Verarbeitung seiner“ Anfragen. „Bis auf gegebenenfalls kurz auftretende Session-spezifische Abbrüche gibt es in der Regel keine größeren Ausfallzeiten, jedoch sinkt die Performance, da die verbliebenden Nodes nun alle Requests bedienen müssen. Eine andere Variante wäre, dass die Nodes verschiedene Ressourcen zur Verfügung stellen, die im Failover-Fall von einem anderen Node übernommen werden.“ [Lie13, *S.224*]. In diesen Fällen existieren mehrere Service-IPs, die verschiedenen Diensten zugeordnet werden. Treten Störungen auf, können diese auf andere Nodes umgeschwenkt werden. Natürlich ist das nur dann sinnvoll, wenn die beteiligten Nodes entsprechend viel freie Kapazität besitzen, beispielsweise eine durchschnittliche Maximallast von 50 Prozent oder weniger.

Die folgenden Abbildungen zeigen ein 2-Node-Active/Active-Setup mit Nodes, die den gleichen Dienst anbieten, vor und nach einem Ausfall.

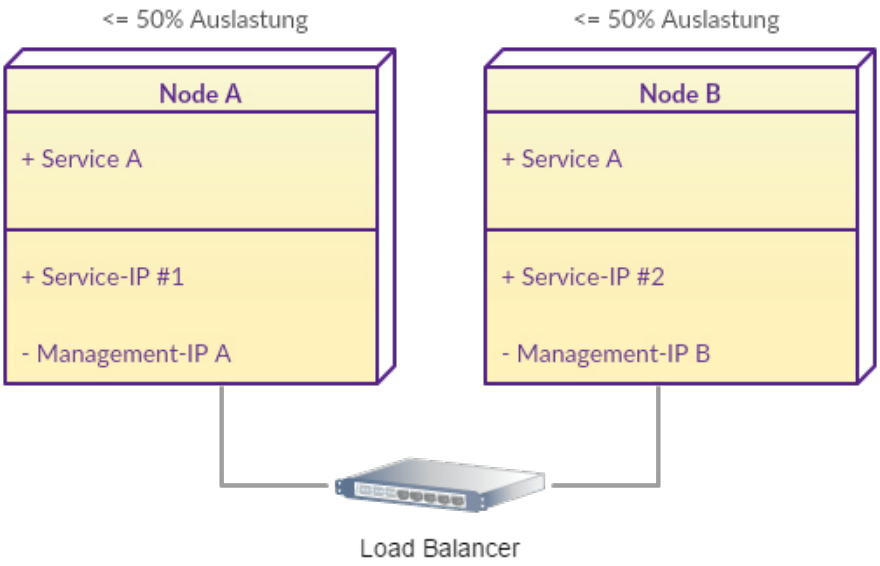

Abbildung 2.5: Active/Active Same Service pre Failover

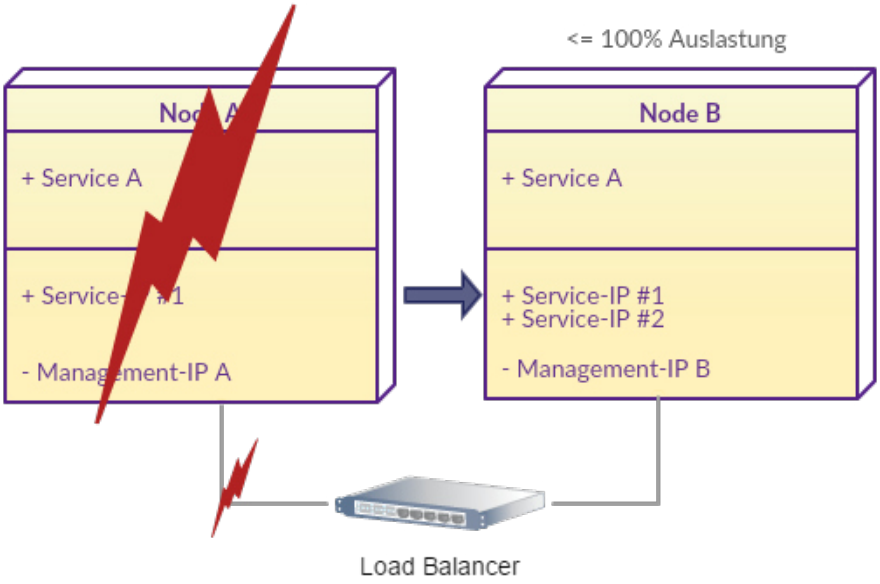

Abbildung 2.6: Active/Active Same Service post Failover

Die Vorteile des Active/Active-Setups sind, dass alle Nodes aktiv genutzt werden und so keine Ressourcenverschwendung vorherrscht. Außerdem können so mehr Dienste gleichzeitig zur Verfügung gestellt werden und der Ausfall eines Nodes „hat primär nur leistungsabhängige Auswirkungen auf den Cluster“ [Lie13, *S.224*]. „Falls multiple Instanzen der gleichen Ressource auf allen Nodes aktiv sind, muss diese Ressource softwaretechnisch auch eine Active/Active-

Konfiguration unterstützen, andernfalls muss sie - gegebenenfalls mit hohem Aufwand - erst entsprechend angepasst werden. Bei einer Active/Passive-Konfiguration hingegen können die verschiedensten Ressourcen ohne Aufwand verfügbar gemacht werden, da der Zugriff auf die Ressource immer nur durch den jeweils aktiven Node erfolgt. Bei verschiedenen Ressourcen auf den Nodes eines Active/Active-Clusters fallen zudem hohe Transitionszeiten durch Service-Starts auf dem jeweiligen Zielnode negativ ins Gewicht. Zudem ist insgesamt eine höhere Anzahl an Transitionen (und damit möglichen Fehlern) [...] erforderlich, da die Services [...] in der Regel wieder auf ihrem angedachten Zielnode laufen sollen.“ [Lie13, *S.224*] Ein Active/Active-Cluster kombiniert also eine effiziente Nutzung vorhandener Ressourcen mit Hochverfügbarkeit, erfordert jedoch sowohl ein tiefergehendes Wissen der angebotenen Dienste als auch der Mechaniken zur Lastverteilung zwischen den Nodes.

Die folgenden Abbildungen zeigen ein 2-Node-Active/Active-Setup mit Nodes, die unterschiedliche Dienste anbieten, vor und nach einem Ausfall.

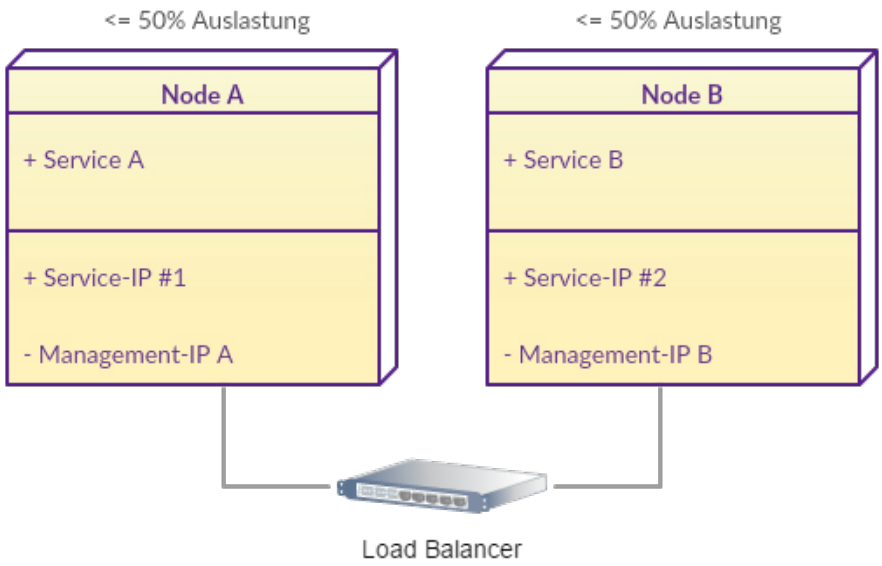

Abbildung 2.7: Active/Active Other Service pre Failover

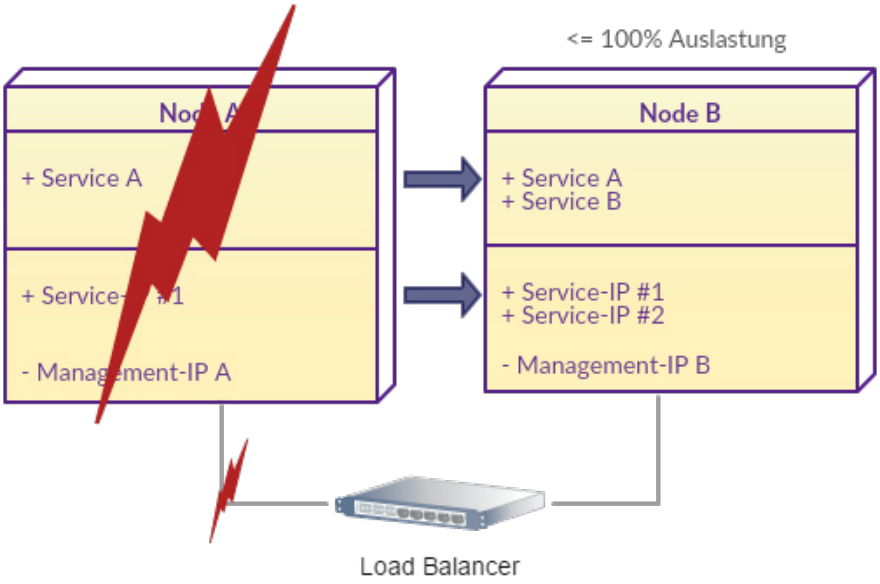

Abbildung 2.8: Active/Active Other Service post Failover

Loadbalancing und Skalierbarkeit

Ein Cluster ist nicht auf zwei Nodes beschränkt, sondern kann theoretisch beliebig viele Teilnehmer verwalten. Aus dieser möglichen Erweiterbarkeit folgt logisch das Prinzip der Skalierbarkeit. Ein bereits hochverfügbares Cluster kann beispielsweise durch die zusätzliche Integration der Open Source Tools HAProxy oder LVS um diese Eigenschaft erweitert werden. Dabei wird zwischen horizontaler und vertikaler Skalierung unterschieden. Erstere steht für das „das Hinzufügen mehrere [sic] logischer Ressourcen-Einheiten zu einem System, die dazu gebracht werden, wie eine einzige Einheit zu arbeiten.“ [BK11, *S.1*]. Dies beschreibt eine Lastverteilung auf zusätzliche Nodes, falls notwendig. Zweitere steht für das dynamische Hinzufügen „von Ressourcen innerhalb einer logischen Einheit“. [BK11, *S.1*]. Also beispielsweise die Erlaubnis mehr Arbeitsspeicher oder einen weiteren CPU-Kern nutzen zu dürfen. Natürlich können und sollten jene Ressourcen wieder abgezogen werden, wenn sie nicht mehr benötigt werden. Um den grundsätzlichen Gedanken der Hochverfügbarkeit dabei nicht durch einen neuen SPoF zu belasten, können die Steuerungsmechanismen in das Cluster selbst integriert werden. Somit kann im Idealfall ein hochverfügbares Servercluster mit dynamischer Skalierbarkeit erreicht werden [Lie13, *S.396ff*].

2.2.3 Ein Blick über den Tellerand: Datenbanken

Eine eigene Teilmenge innerhalb der Redundanz und Failover Thematik stellen die Datenbanken dar. Diese bieten nur eben jenen Dienst an, dessen Leistung sich in Lese- und Schreibzugriff

unterteilen lässt. Dafür ist es um so wichtiger, dass die Datenbestände stets synchron sind. Da Datenbanken üblicherweise mehrmals pro Sekunde, regelmäßig und von verschiedenen Nodes in Anspruch genommen werden, hat sich hier eine etwas abgewandelte Clusterstruktur entwickelt. Es existiert eine Unterteilung in Master/Slave und Master/Master Replikation [Erb12, *S.110f*].

Master/Slave

Bei einer Master/Slave-Konfiguration besteht ein Cluster aus genau einem Master und 1 bis n Slaves. Ein andere Bezeichnung hierfür lautet Passive Replication. Der Master, auch designated primary, führt als einziger Node Schreibzugriffe aus und synchronisiert die Datenbestände daraufhin an die ihm bekannten Slaves. In diesem Zustand des Clusters verteilt ein Loadbalancer die Lesezugriffe, die üblicherweise deutlich häufiger vorkommen als Schreibzugriffe, auf die Slaves. Im Falle eines Failovers übernimmt ein Slave die Rolle des Master und bietet dessen Dienst weiter an. Die Ähnlichkeit zu den allgemeinen Failover-Clustern ist offensichtlich und auch hier spielen ein übergeordneter Cluster-Layer, Rollenmanagement und Kommunikation zwischen den beteiligten Nodes eine elementare Rolle. Bei diesem Ansatz werden die Daten des Master auf die Slaves synchronisiert. Das heißt, nur ein Node hat die Berechtigung Datensätze zu ändern oder zu erstellen. Daraus lässt sich folgern, dass dieser Ansatz wenige Probleme bei Synchronisierung und Cluster Hierarchie aufweist. Ein Loadbalancing der Datenerhebung, also des Schreibens von Daten, ist jedoch nicht möglich [Erb12, *S.110f*].

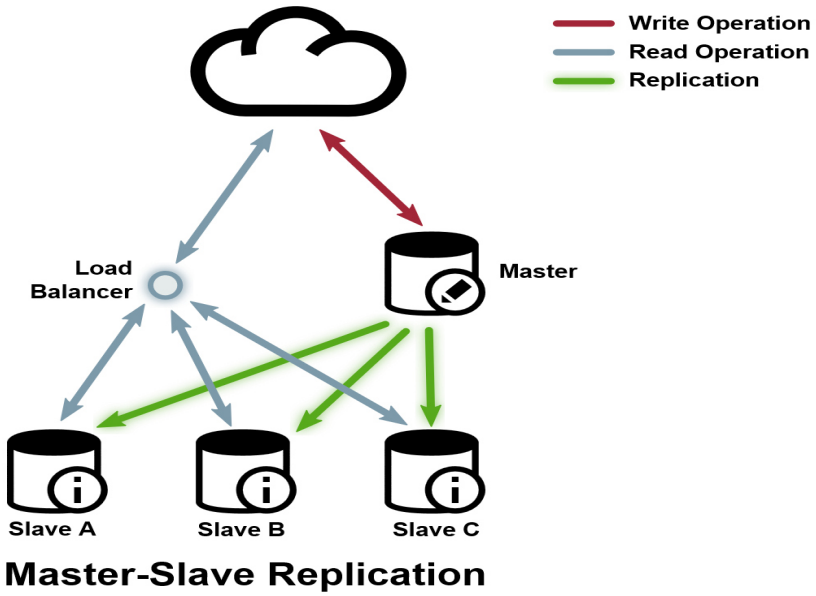

Abbildung 2.9: Datenbank Replikation M/S [Erb12, *S.111*]

Master/Master

Diese Cluster-Struktur, auch Multi-Master Replikation oder active replication genannt, gestaltet sich aufwändiger, da alle Nodes beliebige Befehle erhalten und diese bearbeiten können. Dadurch muss nicht mehr nur unidirektional der Datenbestand des aktuellen Master auf die Slaves synchronisiert werden, sondern alle Master benötigen eine permanente, multidirektionale und konfliktfreie Synchronisation. Das Worst Case Szenario einer Multi-Master Datenbank-Struktur ist die Existenz von unterschiedlichen und gleichzeitig aktuellen Datensätzen. Dabei fehlen beispielsweise Node B Daten, die Node D besitzt und umgekehrt. Es wird also zusätzlich zu der Failover-Struktur ein vollständiges Konfliktmanagement benötigt. Im Ausgleich erhöhen sich Skalierbarkeit und Leistung eines Clusters bei Schreibzugriff ebenso wie alternativ nur bei Lesezugriff [Erb12, *S.110f*].

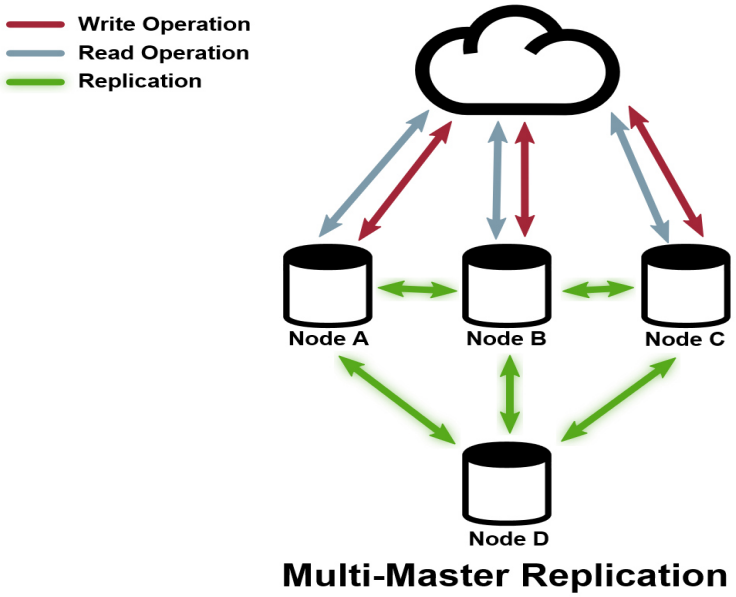

Abbildung 2.10: Datenbank Replikation M/M [Erb12, *S.111*]

2.2.4 Synchronisierung

Abschließend ist für einen automatisierten Failover dynamischer Netzwerkdienste nicht nur der Failover-Prozess an sich relevant, sondern auch, dass der Datenbestand der Nodes möglichst identisch ist. Nur so kann ein Dienst im Falle eines Failover, soweit möglich, auf Node B an der Stelle fortgesetzt werden, an der Node A abgebrochen hat. Dabei existieren vordergründig drei Möglichkeiten einen Datenbestand und Dienst-Status automatisiert zu replizieren, beziehungsweise für alle beteiligten Nodes verfügbar zu machen. Die Replikation des Festplattenspeichers

mittels DRBD und rsync, wobei der Arbeitsspeicher (RAM) außen vor bleibt oder Funktionalitäten, die der betrachtete Dienst selbst mitbringt.

DRBD

Die Software Distributed Replicated Block Device (DRBD) arbeitet mit der Replikation von sogenannten Block Devices oder auch blockorientierten Geräten. Dabei sitzt sie zwischen Dateisystem und dem Treiber der Festplatte. So wird der gesamte Schreibzugriff eines Node A auf einen bestimmten vorher festgelegten Block eins zu eins über das Netzwerk mit einem Node B synchronisiert. DRBD kann als netzwerkbasiertes Raid-1 verstanden werden [LIN].

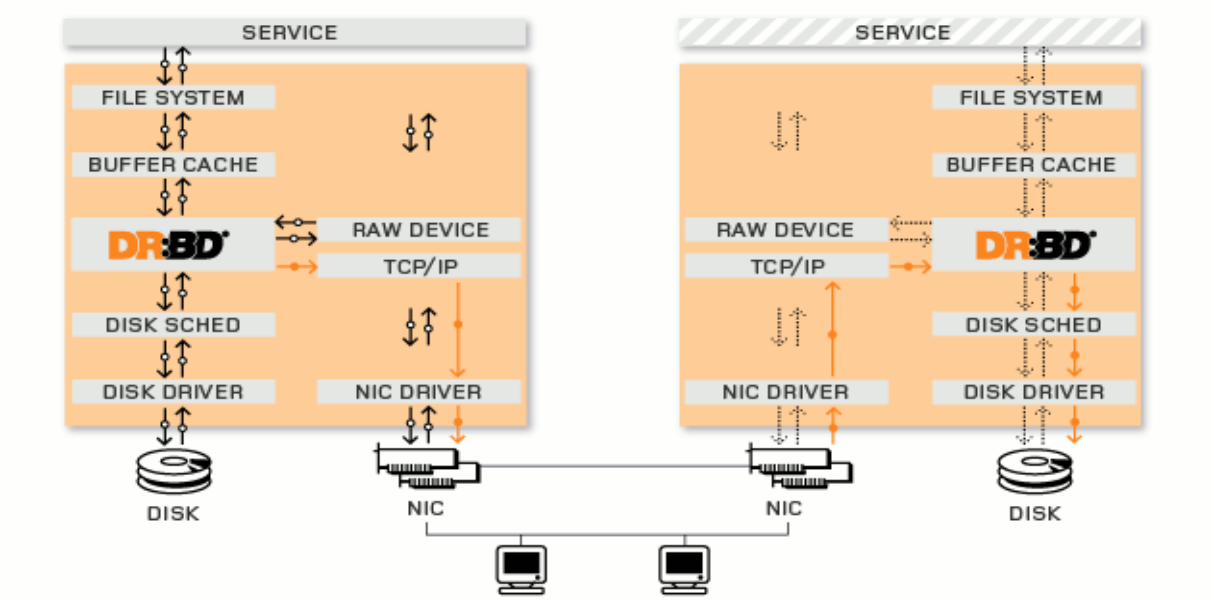

Abbildung 2.11: DRBD [LIN]

rsync

Rsync ist ein Open Source Tool, das inkrementellen Datentransfer ermöglicht. Es werden Änderungen von Dateien innerhalb eines festgelegten Ordners oder einer Partition beobachtet. Tritt eine solche ein, werden ausschließlich die veränderten oder sogar nur der bearbeitete Teil der Dateien synchronisiert. Dies reduziert den Netzwerk-Traffic ebenso wie die Dauer pro Synchronisationsvorgang [Divc].

Eigene Funktionalität

Bei Nutzung von Möglichkeiten der Replikation, die von der abzusichernden Applikation selbst angeboten werden, muss beachtetet werden, dass es sich um eine Lösung nur für diesen einen Dienst handelt. Falls mehrere Dienste angeboten werden sollen, muss der Prozess jeweils wiederholt werden und kann fehlschlagen, falls eine der Applikationen keine solche Funktionalität anbietet. Die Bandbreite solcher Funktionen reicht von simplem Logging über das Ex- und Importieren von Daten bis hin zu Multi-Instanz-Management der Dienste untereinander.

2.2.5 Netzverkehr und Routing

Um alle Aspekte eines Failover-Clusters behandeln zu können, muss auch festgelegt werden in welcher Form und auf welche Art und Weise Daten die Nodes erreichen. Es folgt eine Übersicht zu Arten der Datenübertragung in paketvermittelnden Netzen.

Unicast

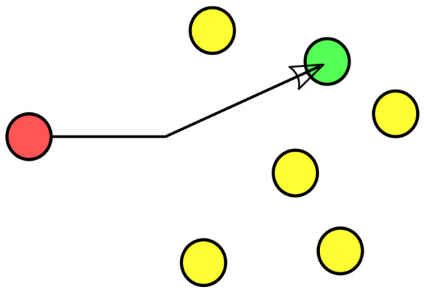

Abbildung 2.12: Unicast, nach [Eas06, *Abbildungen*]

Unicast entspricht dem klassischen Transport in paketvermittelnden Netzen. Ein Sender adressiert ein Paket an einen Empfänger [Cis01, *S.571*]. Redundanz kann dabei von den Adressaten selbst realisiert werden. In einem Failover-Szenario beansprucht Node B die Unicast-Adresse, welche vorher Node A innehatte. Gesteuert werden kann dies mittels Kommunikation der Nodes selbst, womit die Generierung eines neuen SPoF vermieden wird. Es besteht Kompatibilität zu Active/Passive-Clustern sowie zu Active/Active-Strukturen mit oder ohne Loadbalancer.

Anycast

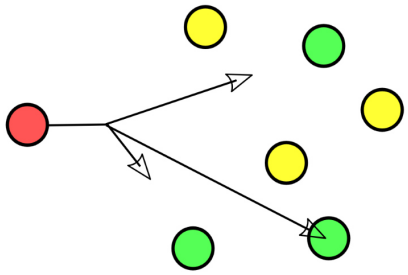

Abbildung 2.13: Anycast, nach [Eas06, *Abbildungen*]

Eine Alternative zum gewöhnlichen Loadbalancer ist die Verwendung von Anycast. Dabei wird das (Open) Shortest Path First Prinzip (OSPF) angewandt. Das heißt, dass je nach Abstraktionsebene alle Cluster oder alle Nodes die gleiche IP-Adresse reservieren, dazu jedoch einen Wert mitliefern, wie teuer die von ihnen angebotene Route ist. Umso mehr Vermittlungsstellen zwischen Sender und Empfänger liegen, desto teurer wird der Pfad. So werden Datenpakete stets zum nächstliegenden Empfänger gesendet [Gro, *1546*]. Anycast eignet sich also für das globale Ansteuern von Clustern, jedoch weniger für Hochverfügbarkeit. Dies ist darin begründet, dass ein Loadbalancer die Verteilung der Last intelligenter löst und die Instanz, welche als Anycast-Sender fungiert, einen neuen SPoF darstellt.

Multicast

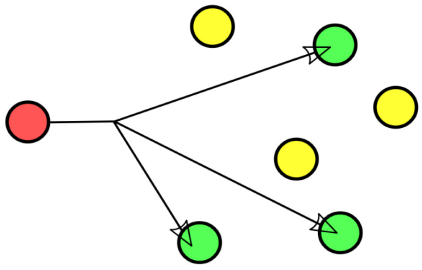

Abbildung 2.14: Multicast, nach [Eas06, *Abbildungen*]

In einem Multicast Netzwerk werden Daten an mehrere Nodes gleichzeitig gesendet [Gro, *1112*]. Im Kontext eines Failover-Cluster bedeutet dies, dass im Idealfall mehrere aktive Nodes die gleichen Anfragen bearbeiten, jedoch nur ein Node antworten würde. Da die weiteren

Nodes ebenso alle darauf folgenden Pakete erhalten, bestehen im Falle eines Failovers sogar im RAM die gleichen Zustände wie bei Node A. Außerdem ist Node B bereits aktiv und bietet dem Administrator somit eine minimale Transferzeit. Erhebliche Nachteile sind jedoch, dass doppelte Kosten und Wartungsarbeiten ohne jeglichen produktiven Mehrwehrt entstehen und, dass Systemfehler, die durch einen Angriff oder kritische Datenpakete verursacht werden, sowohl den aktiven als auch den redundanten Node beeinträchtigen könnten.

2.3 Existierende Lösungen

Komplettlösungen und Methoden zur Bewältigung der Failover-Thematik existieren in Form von speziell hierfür entwickelter Hard- oder Software, die entsprechende Erweiterungen und Funktionen anbietet. Diese Arbeit beschränkt sich aufgrund der praktischen, Node-bezogenen Aufgabenstellung auf die Betrachtung von Softwarelösungen. Des Weiteren kann das Angebot in kommerzielle und freie Angebote unterteilt werden. Eine primäre Motivation für die Verwendung von freien Angeboten besteht in den Kosten einiger kommerzieller Lösungen und deren Support. Hinzu kommt das möglicherweise bereits vorhandene Linux Know-How in einem Unternehmen. Einige spezialisierte Betriebssysteme für Hochverfügbarkeit basieren zwar auf Linux, jedoch kann in diesem Fall der Heimvorteil einer IT-Abteilung genutzt werden. So kann diese fehlende Elemente selbst hinzufügen und Fehler beheben, ohne stets auf einen externen, kostenpflichtigen Support zurückgreifen zu müssen. Außerdem existiert für Open Source Projekte meist eine große Community, die aktiv mitentwickelt, Support leistet und sich effektiv in Projektgruppen aufteilen kann. So steigen Wartungsqualität und Weiterentwicklung mit jedem Anwender.

2.3.1 Marktsituation

Der global agierende Konzern für Unternehmenssoftware SAP beispielsweise hat seinen Umsatz im Bereich der Cloud Dienste vom dritten Quartal 2014 hin zu gleichem Quartal im Jahre 2015 um 116% steigern können [Pre]. Dabei geht SAP Vorstandssprecher Bill McDermott, Stand Oktober 2015, auf die Bedeutung der Cloud Dienste ein: „Unsere HANA Cloud Plattform für das Internet der Dinge und unsere Geschäftsnetzwerke definieren den Markt für Unternehmenssoftware neu.“ [Pre]. Dazu kommen die vielen Dienstleister, von Telekommunikationsunternehmen bis hin zu Börsen und Anbietern von Applikationen für mobile Geräte und Computer, die ihre Dienste in Gänze oder in Teilen in die Cloud auslagern. Ist einer der beteiligten Server nicht erreichbar, kann die erwartete Leistung nicht mehr abgerufen werden. Dabei interessieren den Kunden weniger die Gründe, Dauer und Häufigkeit der Ausfälle dafür um so mehr.

Unter anderem deshalb rücken Themen wie automatisierte Ausfallsicherungen, Failover und HA immer mehr in den Fokus von IT-Abteilungen und Netzwerkspezialisten. Eine Verfügbarkeit von 99 Prozent oder höher wird zwischen Unternehmen meist bereits innerhalb der SLA bei Vertragsabschluss festgelegt. Einige Produkte beinhalten diese Garantien auch für Privatkunden. Unterschiede bei den Güteklassen zeigen sich primär innerhalb der letzten fünf Prozent, da die garantierte Verfügbarkeit meist zwischen 95,0 und 99,999 Prozent variiert. Auch dieses Zahlenspektrum kann einen großen Unterschied bezüglich der erlaubten Ausfallzeiten bewirken. Die Deutsche Telekom hat einer Veröffentlichung ihrer Zahlen und SLA an dieser Stelle nicht zugestimmt.

Zur Verdeutlichung des Genannten sollen einige Daten zu der garantierten Verfügbarkeit des Server Anbieters Host Europe aufgezeigt werden, welche dieser öffentlich zugänglich präsentiert. Auf seiner Homepage bewirbt der Dienstleister seine Webhosting Services mit einer Verfügbarkeit von 99,9 Prozent. Bei Anmietung eines Root Servers werden immer noch 99,95 Prozent garantiert. Dem der Arbeit beigefügten Service Level Agreement lässt sich entnehmen, dass zusätzlich noch weitere Vereinbarungen getroffen werden können [Eur15, *S.3*].

Telekommunikationsanbieter der Ebenen eins und zwei (Tier) garantieren häufig eine Verfügbarkeit von 99,9 bis 99,999 Prozent (Carrier Grade), wobei letztere auch als *Five Nines* bezeichnet wird [Col04, *S.2*]. Folgende Abbildung verdeutlicht, welcher marktwirtschaftlicher Verlust für einen Anbieter entstehen kann, falls die festgelegten Werte nicht eingehalten werden. Selbst bei der kleinsten garantierten Verfügbarkeit bewirken drei Prozent Ausfallzeit zu viel bereits den vollständigen Verlust der Einnahmen für das Produkt.

Garantie	**99,999%**	**99,995%**	**99,99%**	**99,95%**	**99,9%**	**Gutschrift**
Verfügbarkeit	<99,999%	<99,995%	<99,99%	<99,95%	<99,9%	5%
	<99,998%	<99,991%	<99,97%	<99,91%	<99,8%	10%
	<99,995%	<99,981%	<99,94%	<99,81%	<99,6%	25%
	<99,991%	<99,963%	<99,89%	<99,63%	<99,3%	50%
	<99,981%	<99,926%	<99,78%	<99,26%	<98,5%	75%
	<99,963%	<99,852%	<99,56%	<98,52%	<97,0%	100%

Tabelle 2.4: Host Europe SLA Verfügbarkeit, nach [Eur15, *S.8*]

Darüber hinaus garantiert Host Europe „eine Verfügbarkeit der Internetverbindung von 99,999 Prozent im Monatsmittel“ [Eur15, *S.6*]. Es wird also sichtbar, dass ein Glied einer Dienstleistungskette maximal die Verfügbarkeit anbieten kann, die das nächst höhere anbietet. Bei einem SLA mit 99,999 prozentiger Verfügbarkeitsgarantie und einer eigenen Vertragsleistung von ebenfalls 99,999 Prozent durch den Internetprovider bleibt kein Puffer für Host Europe selbst. Dabei entsprechen die prozentual gering wirkenden Unterschiede zeitlich durchaus be-

achtlichen Volumina. Unter der Annahme, dass ein Jahr 365 Tagen entspricht, ergeben 99,99 Prozent maximal 52,56 Minuten Ausfallzeit pro Jahr. Dies ergibt 4,38 Minuten pro Monat. Die Zeiträume zu 99,999 Prozent, also Carrier Grade, belaufen sich auf 5,256 Minuten pro Jahr oder ungefähr 0,43 Minuten pro Monat. Es wurde in diesem Fall abgerundet, da ein Aufrunden bereits eine Verletzung der Garantie darstellen würde. Werden diese Werte nicht eingehalten, drohen Vertragsstrafen.

Unter anderem nutzt auch die Kerninfrastruktur der Tier-1 Telekommunikationsanbieter sogenannte Signaling System No. 7 (SS7) Netze. Diese garantieren Verfügbarkeit durch den Einsatz von Mated Signaling Transfer Points (STP). Es handelt sich also um paarweise angeordnete und vernetzte STP. Diese arbeiten mit einem Aktiv-Aktiv Redundanz Prinzip, wobei zwei Knoten jeweils die gleiche Aufgabe erfüllen und an möglichst entgegengesetzten Orten eines Staates stationiert sind [Sta04, *S.90*]. Jener Aufbau kann bewirken, dass ein Paket an einen STP in Stadt A geroutet wird, die Antwort jedoch von einem STP aus einer Stadt B gesendet wird.

2.3.2 Kommerzielle Lösungen

Anbieter

Zu den Anbietern kommerzieller Failover- und Loadbalancing Software zählen Global Player wie Cisco, HP, Microsoft und Oracle ebenso wie Entwickler speziell angepasster Linux Distributionen. Einige Namen deren HA Angebote lauten in gleicher Reihenfolge wie folgt: Integrated Services Routers, Serviceguard, Windows Server, Solaris Cluster und, zum Beispiel, Red Hat Enterprise Linux.

Preismodelle

Alle oben genannten Unternehmen, die Softwarelösungen in ihrem Portfolio verzeichnen, setzen auf ähnliche Preismodelle und orientieren sich dabei an Volumenlizensierung. Konkret wird sich auf die Anzahl vorhandener CPU-Sockel oder Systeme bezogen. Während einige ihr zugrundeliegendes Betriebssystem (OS) kostenlos anbieten, wie Oracle seine Linux Distribution [Oraa], sind Support sowie Nutzung von HA und Failover Mechaniken stets kostenpflichtig [Orab; Orac]. Microsoft und andere Mitbewerber nennen die Preise ihrer Lösungen nur auf Anfrage. Daraus könnte gefolgert werden, dass jene Kosten im ersten Moment abschreckend hoch erscheinen und daher erst in einem aktiven Sales-Prozess kommuniziert werden. Es ist bei allen aufgeführten Angeboten der Fall, dass die Preise, je nach gewünschtem Support und Größe der Cluster, individuell festgelegt werden.

Red Hat etwa verlangt für sein Betriebssystem Red Hat Enterprise Linux (RHEL) zur Zeit 345,50 USD/Jahr. Oracle bietet sein OS zwar kostenlos an, verlangt für sein Solaris Cluster jedoch 1000 USD/Jahr pro Sockel. Beide Beispiele beinhalten Support und gelten für einen Sockel eines x86 Servers. Dabei handelt es sich bei Oracle um Premium und bei Red Hat um Standard Support. Es folgt eine Beispielrechnung für Produkte der beiden oben genannten Anbieter bezogen auf ein Jahr. Gegeben sei ein kleines Unternehmen, das vier x86 Server mit jeweils zwei Dualcore CPUs betreiben möchte. Genannte und folgende Zahlen basieren auf [Oraa] sowie [Hat], also offiziellen Angaben der Anbieter.

Produkt	**Menge**	**Preis pro Sockel**	**Gesamtpreis pro Jahr**
RHEL Server Standard Support	8	399,50 USD	3.169,00 USD
RHEL Server Premium Support	8	649,50 USD	5.196,00 USD
RHEL VDs Standard Support	8	999,99 USD	10.392,00 USD
RHEL VDs Premium Support	8	1.624,50 USD	7.996,00 USD
Oracle Solaris Cluster	8	1.000,00 USD	8.000,00 USD
Oracle Network Support	4	119,00 USD	476,00 USD
Oracle VM Premier Support	4	1.199,00 USD	4.796,00 USD
Oracle Linux Premier Support	4	6.897,00 USD	27.588,00 USD

Tabelle 2.5: Preise an einem Beispiel

Dabei beschränken sich die Support Lizenzen bei beiden Anbietern auf jeweils ein System, beinhalten für dieses jedoch bereits eine Nutzungslizenz. Ein System entspricht hierbei einem Server mit beliebiger Anzahl Sockel. Es ist also möglich, dass vor und während der Unterstützung eine Prüfung der ID des Systems stattfindet und gegebenenfalls ausschließlich für dieses Support geleistet wird. Außerdem fällt auf, dass beide Lösungen auf Linux basieren.

2.3.3 Freie Lösungen

Da die im vorherigen Abschnitt näher betrachteten Failover Lösungen auf Linux basieren, stellt sich für ein kostenorientiertes Unternehmen mit einer Linux-erfahrenen IT die Frage, ob ähnliche Ansätze im Jahr 2015 nicht auch frei und quelloffen existieren. Diese Frage kann positiv beantwortet werden. Dabei gibt es diese nicht nur als fertige, spezialisierte Betriebssysteme, sondern ebenso in Form von Modulen und Paketen für verschiedene Linux Distributionen. Die erste, grundlegende Version von Linux-HA erschien bereits am 15. November 1998 [Sch12, *S.24*]. Zu diesem Zeitpunkt enthielt diese Lösung bereits eine Service-IP-Adresse, Heartbeat und weitere Features. Bis zum Jahre 2015 kann Linux-HA auf stetige Weiterentwicklung zurückblicken [Sch12, *S.25*]. „Das monolithische heartbeat bekam mit der Version 2

intern schon eine modulare Struktur. Der Cluster Manager pacemaker konnte sowohl OpenAIS als auch heartbeat für die Kommunikation im Cluster nutzen. Bis Version 1.05 von pacemaker musste das komplette heartbeat-Paket mit installiert werden, ab dieser Version genügen der cluster-glue und die resource-agents.“ [Sch12, *S.28*]. *Monolithisch* steht in Elektronik dafür, dass ein Ganzes untrennbar aus diversen kleinen Elementen besteht und diese nicht variabel sind [Dud, *monolithisch*].

Folgende Abbildung bietet einen Überblick über die Evolution der Linux Cluster-Software in den letzten Jahren.

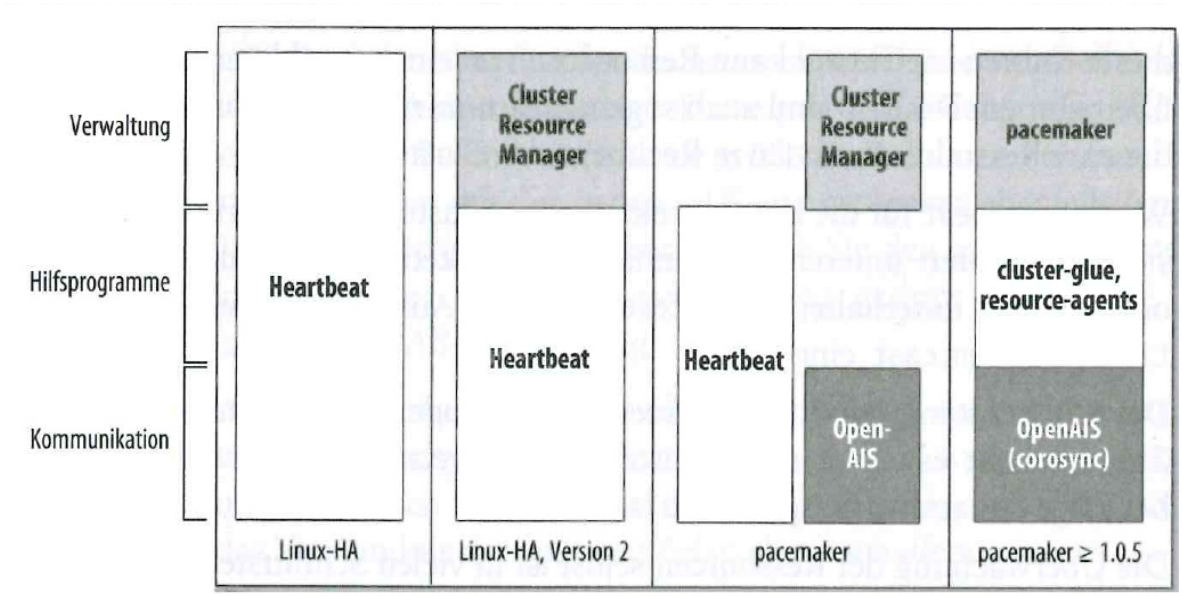

Abbildung 2.15: Linux-HA Geschichte [Sch12, *S.29*]

Heartbeat - Version 1

Heartbeat stellte lange das Herzstück des Linux-HA Gedanken dar. Grundsätzlich steht diese Lösung, wie der Name schon sagt, für sogenannte Heartbeats: „Ein Cluster bestand aus zwei Rechnern, die sich wechselseitig Statusmeldungen [...] zuschickten. Auf dem aktiven Rechner im Cluster wurden die Dienste gestartet. Der andere Rechner war in dieser Version rein passiv und sprang nur im Notfall ein. Dieser Fall trat dann ein, wenn die Statusmeldungen des aktiven Systems eine Zeit lang ausblieben. Der bis dahin passive Rechner nahm daraufhin an, der erste Rechner habe ein Problem bekommen und startete die Dienste. Als Dienst zählte zum Beispiel auch die IP-Adresse, unter der der Cluster als Gesamtsystem erreichbar war.“ [Sch12, *S.25*]. Das klassische Heartbeat entsprach also einem 2-Node-Active/Passive-Cluster.

„Die Konfiguration eines Clusters entsprechend Version 1 von Linux-HA bestand aus drei Dateien: In der ersten Datei authkeys wurde ein gemeinsames Passwort hinterlegt, sodass diese Kommunikation zwischen den Knoten abgesichert war.“ [Sch12, *S.25*]. Dies gleicht der Authentifizierung innerhalb des SNMP-Protokolls, bei der ebenfalls lediglich ein Passwort, Community Name genannt, zur Absicherung festgelegt wird. „Die zweite Datei ha.cf beinhaltete die Einstellungen, die das Verhalten des Cluster als Gesamtes betreffen. Hier legte der Administrator fest, welche Rechner Mitglied des Clusters waren, wie die Kommunikation zwischen ihnen funktionierte, und zum Beispiel auch, wie lange die Meldungen des anderen Rechners ausbleiben durften, bis der Fehlerfall ausgerufen wurde. In der dritten Datei haresources definierte der Administrator schließlich, welche Dienste, auch Ressourcen genannt, der Cluster verwalten sollte. Weil die heartbeat-Software in Version 1 monolithisch aufgebaut war, lief sie stabil und wurde in vielen Installationen eingesetzt. Der Nachteil des monolithischen Aufbaus war, dass die ganze Architektur statisch war und damit ziemlich unflexibel. Ressourcen liefen immer auf einem Rechner, und es war unmöglich, sie einfach auf den anderen Rechner umzuschwenken. Kurz, es fehlte ein vollständiger Cluster-Manager, der die Ressourcen verwaltete.“ [Sch12, *S.25*]. „Ein zusätzliches Problem war, dass ein Cluster entsprechend der Version 1 den Zustand der Ressourcen selbst nicht überprüfen konnte. Die einzige Überprüfung im Cluster war der Austausch der Heartbeats, der nur etwas über den Zustand der Cluster-Software bzw. des Knoten selbst aussagte. Der Cluster wusste nichts über den Zustand der Ressourcen. Er selbst konnte nicht reagieren, wenn der Webserver abgestürzt war und keine Seiten mehr auslieferte [...]. Für solche Tests waren zusätzliche Programme notwendig [...].“ [Sch12, *S.25*]. Komplexe Prüfungen der Verfügbarkeit auf Applikationsebene oder mittels mehrerer, verschiedenartiger Tests je Dienst sind in dieser Version also nicht oder nur durch den Einsatz externer Software möglich.

Heartbeat - Version 2

„Die oben angesprochenen Nachteile gaben den Anstoß zur Entwicklung der Version 2 von Linux-HA [...]. Ein eigener Cluster-Manager kümmerte sich um die Ressourcen. Er konnte selbst die Ressourcen auf ordnungsgemäße Funktion hin überprüfen und auf Fehler der Ressourcen reagieren, sie also zum Beispiel auf einem anderen Knoten starten. Externe Programme zur Überwachung waren nicht mehr notwendig. Der Administrator konnte mit dem neuen Cluster Resource Manager (CRM) komplexe Zusammenhänge zwischen den Ressourcen eingeben, sodass unterschiedliche Ressourcen auf unterschiedlichen Knoten laufen konnten. Diese Fähigkeit war besonders interessant, da ein Cluster nicht mehr nur aus zwei Knoten bestehen konnte, sondern bis zu 16 Rechner einen Cluster bilden konnten.“ [Sch12, *S.26*]. „Alle Rechner waren prinzipiell gleichberechtigt. Über die Konfiguration des Clusters konnte der Administrator die Entscheidung des CRM dahin gehend beeinflussen, welche Ressourcen wo laufen sollten [...].“ [Sch12, *S.26*]. Jene Weiterentwicklungen von Heartbeat ermöglichen also Strukturen mit bis

zu 16 Nodes, Active/Active Cluster, grundsätzliches Loadbalancing und das direkte Einbinden von Ressourcen mittels Resource Agents, womit eine Verfügbarkeitsprüfung auf Applikationsebene, OSI Schicht (5 bis) 7, denkbar ist. Das ermöglicht eine Beobachtung aller kritischen Komponenten eines Dienstes und dies logisch gebündelt in einer Ressource mit zugehörigem ressource-agent [Lie13, *S.459;S.326*]. Es wird empfohlen, zu dem Open Cluster Framework (OCF) kompatible Agenten zu verwenden, welche bei einer Installation von Heartbeat enthalten sind oder von entsprechenden Github-Quelle bezogen werden können. Weiterhin ist es möglich, sogenannte Linux Standard Base (LSB) Scripte einzubinden, die in der Regel für die allgemeine Steuerung von Diensten unter Linux verwendet werden.

„Da die Konfiguration nun dynamisch während des Betriebs geändert werden konnte, um Ressourcen zwischen den Rechnern hin- und herzuschieben, haben die Entwickler die starre Konfiguration mithilfe der Textdatei haresources aufgegeben und diese in eine sogenannte Cluster Information Base (CIB) verlegt. Diese Konfigurationsdatei verwaltete der Cluster selbst dynamisch und der Administrator kann sie nicht mehr direkt editieren. Die ganze Kunst des Clusterbaus entsprechend Version 2 von Linux-HA bestand für den Administrator darin, seine Wünsche in der CIB für den Cluster verständlich zu hinterlegen.“ [Sch12, *S.26f*]. „Ab Version 2.99 von heartbeat [sic] hat die Software den CRM nicht mehr integriert und kann deshalb zusammen mit pacemaker für eine hohe Verfügbarkeit der Dienste sorgen.“ [Sch12, *S.27*].

Pacemaker

„Aufgrund interner Streitigkeiten haben die Entwickler des Cluster-Managers ihre Software aus dem Linux-HA-Projekt herausgelöst [...]. Die zentrale Schaltstelle im Cluster heißt nun pacemaker.“ [Sch12, *S.26f*]. Die Grundidee zur Konfiguration blieb dabei dieselbe. „Als Erstes änderten die Entwickler die Schnittstelle, die der Manager für die Kommunikation im Cluster benötigte. Neben heartbeat kann pacemaker deshalb auch OpenAIS nutzen, um Nachrichten zwischen den Knoten weiterzuleiten und sich über die Mitgliedschaft von Rechnern im Cluster berichten zu lassen.“ [Sch12, *S.26f*]. Nach der modularen Auslegung der zweiten Heartbeat Version, obliegt es ab diesem Entwicklungsstand zusätzlich dem Anwenders, welcher Unterbau des Cluster-Managers Pacemaker für die Kommunikation innerhalb des Clusters und die Verfügbarkeitskontrolle angewandt werden soll. Außerdem ist herauszustellen, dass es sich bei Pacemaker um ein dezentral gesteuertes Cluster-Gehirn handelt, das aufgrund seiner Verteilung keinen neuen SPoF generiert.

„Zusammen mit Version 1.0.5 [...] haben die Entwickler die verbliebene Software von Linux-HA noch einmal aufgeräumt und sie in drei verschiedene Pakete aufgeteilt:

resource-agents: In diesem Paket sind alle Agenten zusammengefasst. Die Agenten bilden das Bindeglied zwischen der Clustersoftware und den tatsächlichen binären Programmen, die

der Cluster ausführen soll. Sie sind vergleichbar mit den Skripten, die das init-System nutzt.
cluster-glue: Hier finden sich alle Programme aus dem heartbeat-Paket, die auch für den Betrieb eines pacemaker/OpenAIS-Clusters noch notwendig sind.
heartbeat: Ab Version 3 von heartbeat [sic] ist in diesem Paket der Rest der ursprünglichen Software zusammengefasst, die nicht in den anderen beiden Paketen benötigt wird.“ [Sch12, *S.27*].

OpenAIS

„Ursprünglich hatte das OpenAIS-Projekt den Anspruch, alle Vorgaben der Application Interface Spezification (AIS, daher auch der Projektname) des Service Availability Forum (SAF) abzubilden.“ [Sch12, *S.28*]. „pacemaker nutzt die Funktion von OpenAIS [...] nicht vollständig, sondern nur die Bereiche Messaging und Membership. Teilweise bietet OpenAIS auch die gleichen Management-Funktionen wie pacemaker. Deshalb darf OpenAIS in diesem Einsatz seine Fähigkeiten nicht ganz ausspielen [...]. Aus diesem Grund haben die Entwickler von OpenAIS die für ein pacemaker-Cluster notwendigen Funktionen in einem eigenen Projekt corosync zusammengefasst. Diese Software bietet dann die Infrastruktur, auf der ein pacemaker-Cluster aufsetzen kann.“ [Sch12, *S.28*]. Der Cluster-Manager Pacemaker kann für das Grundgerüst seiner Kommunikation also auf die aktuelle Version von Heartbeat oder das komplexere Corosync in der Version 1 oder 2 zurückgreifen.

Keepalived

Keepalived ist eine Hochverfügbarkeits- sowie Loadbalancinglösung und setzt auf der Software Linux Virtual Server (LVS) auf, bei der mehrere physikalische zu einem virtuellen Server zusammengeschlossen werden [Wen98]. Der Fokus beider Projekte liegt auf Loadbalancing, sie bieten jedoch ebenfalls Verfügbarkeitsprüfungen auf den OSI-Schichten 3, 4 und 5 an [Cas]. Dabei kann Keepalived auch um selbstgeschriebene Verfügbarkeits-Checks erweitert werden, in Form sogenannter *Misc Checks* [Cas02, *S.7*]. Die Failover-Funktionalität wird mittels VRRP erreicht. Laut dem initialen Autor von Keepalived handelt es sich um ein Paket, dass die Funktionen der sonst einzeln erhältlichen Module Heartbeat, LVS und weiterer kombiniert [Cas].

Weitere

Abgesehen von den fortgeschrittenen Entwicklungen des Linux-HA Projekts existieren noch weitere freie Ansätze und Projekte die teils unterschiedliche Prioritäten und Fähigkeiten besitzen. Vier solcher Projekte die mehr oder weniger komplexe Failoverstrukturen innerhalb eines Clusters berücksichtigen sind BeeGFS, Ceph, Tahoe-LAFS und XtreemFS. Ceph sei hier ge-

sondert aufgeführt, da diese Open Source Software den neuartigen Ansatz des Object Storage implementiert. Dabei werden abgelegte Daten in Objekte verpackt, die nach einem Verteilungsalgorithmus, genauer dem CRUSH-Algorithmus, auf verschiedene Gruppen innerhalb des Clusters verteilt werden. So existiert jede Datei redundant auf mindestens zwei Nodes, jedoch nicht auf allen. Weitere Projekte sind das bereits erwähnte LVS sowie HAProxy, das Hochverfügbarkeit bietet und primär für Web-Server mit hohem Aufkommen von Netzverkehr und Loadbalancing entwickelt wurde [hap].

3 Konzeption der Lösung und ihrer Auswertung

3.1 Das Ziel

Ein grundlegendes Ziel besteht darin, die Überwachung der Verfügbarkeit von dynamischen Netzwerkdiensten mit Open Source Software zu realisieren. Die Anforderungen sind dabei besonders bezüglich des Praxisbeispiels Kamailio SIP Proxy so definiert, dass die Überwachung auf höchster Ebene stattfinden soll und somit auf der Applikationsschicht. Es wird nicht nur relevant sein, ob ein Node überhaupt reagiert, wie beispielsweise bei einem Ping oder der Version Eins des Linux Heartbeat, sondern es soll ebenfalls eine Plausibilitätsüberprüfung der Antworten stattfinden. Hierdurch kann später von garantierter Verfügbarkeit der beteiligten Nodes sowie des Dienstes an sich ausgegangen werden.

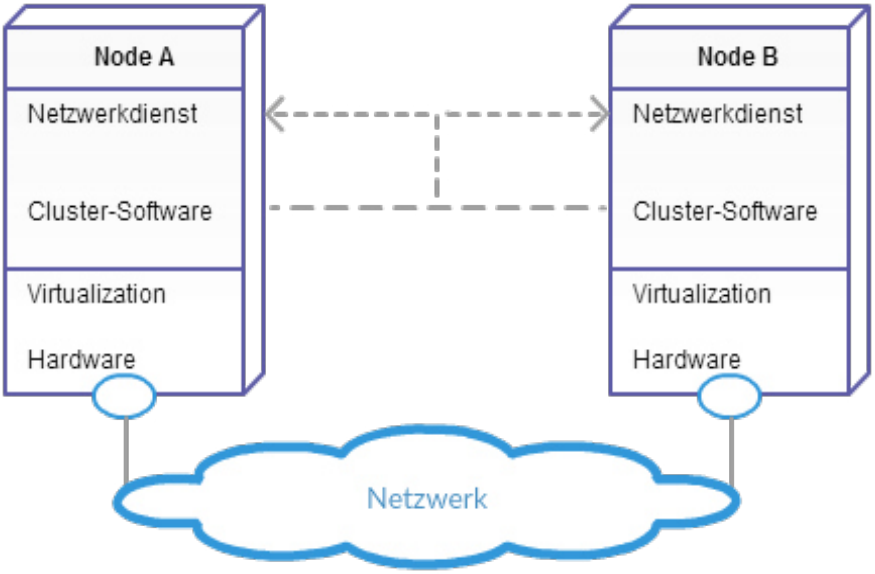

Abbildung 3.1: Grundstruktur

Ein weiteres Teilergebnis stellt das Signal dar, das von der ausgewählten Lösung ausgegeben wird, wenn ein Node nicht mehr verfügbar ist. Jenes ist idealerweise abgreifbar, um die Überwachung der Verfügbarkeit für verschiedene Szenarien und Cluster-Strukturen nutzbar zu machen. Die ausgelöste Menge von Aktionen kann so beliebig variiert und den Anforderungen angepasst werden.

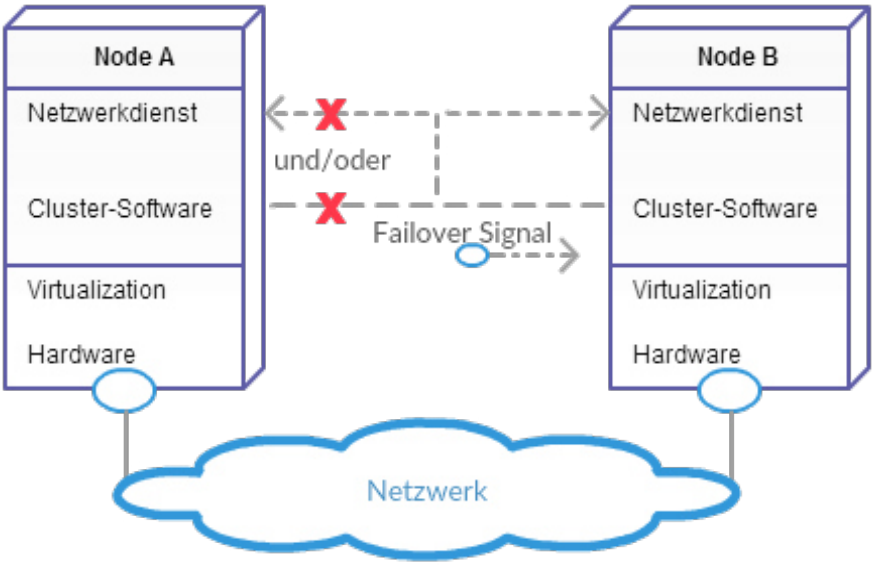

Abbildung 3.2: Grundstruktur mit Signal

Es folgt der eigentliche Failover-Prozess, also die automatische Übernahme eines Dienstes durch einen weiteren Node mit möglichst geringen Auswirkungen für den Dienstnehmer. Ist dieser Meilenstein erreicht, soll der Fokus auf der Erhaltung von dynamischen Status und deren Relevanz für einen Failover gelegt werden. Bei genauerer Betrachtung der UML-Konzeptionen fällt auf, dass in einem Cluster mit genau zwei Nodes nach Eintritt eines Failovers die Cluster-Software des Nodes B für die Überwachung der Verfügbarkeit von Node B zuständig ist. Schematisch sollten Überwachung sowie Messung des Failover-Prozesses auf Erfolg die Aufgabe eines weiteren im Cluster befindlichen Nodes sein. Es kann offensichtlich keine ausreichende Prüfung der Verfügbarkeit eines Netzwerkdienstes stattfinden, wenn Node B bestätigt, dass ein Dienst auf Node B erreichbar ist. Daher wird an dieser Stelle ein Node C eingeführt, dessen einzige Aufgabe die Überwachung der Erreichbarkeit der anderen Nodes ist. In einem produktiven Umfeld würde dieser Node voraussichtlich zusätzliche Funktionen erfüllen. Für weitere Informationen zu beachtenswerten Aspekten nach einem Ausfall siehe auch die Abschnitte 2.1.9 und 2.1.10.

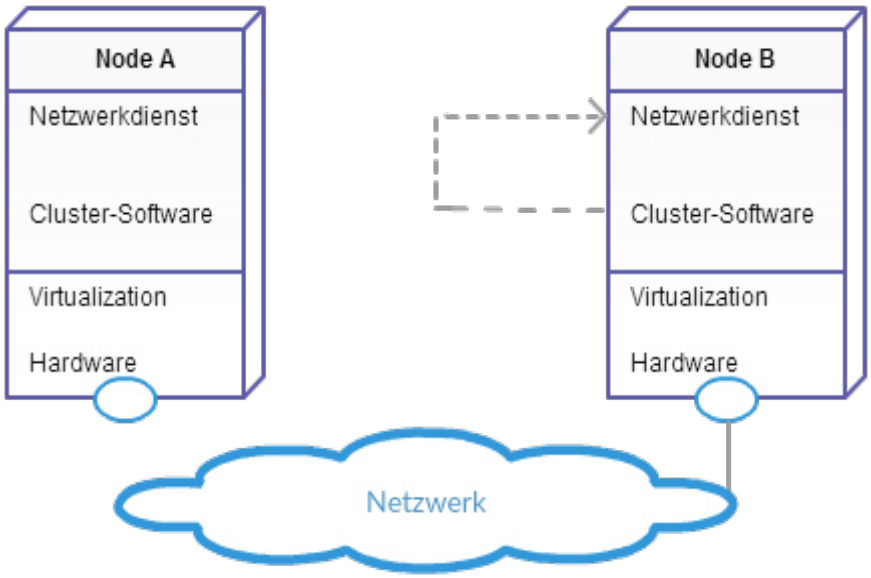

Abbildung 3.3: Grundstruktur nach Failover

Zur Fortführung der Konzeption werden die aufgeführten Thematiken nun in obligatorische und optionale Bedingungen unterteilt. Ob ein Failover erfolgreich verläuft, ist letztendlich mittels Vergleich von empirischen Messwerten mit den Zielwerten der folgenden Tabellen zu bestimmen.

3.2 Abgrenzung der Thematik

Je nach gewähltem Cluster-Prinzip kann ein Netzwerk diverse, unterschiedliche und gleichartige Komponenten sowie Knotenpunkte enthalten. Dazu zählen der Internet Service Provider (ISP), die Firewall, die Server selbst sowie Loadbalancer, diverse Router, Switches und eben solche Geräte, die Verteilungsaufgaben in einem Cluster übernehmen. Im Realfall müssen diese ebenfalls redundant und ausfallsicher vorgehalten werden, um Hochverfügbarkeit garantieren zu können. Die folgende Abbildung 3.4 zeigt einige offensichtliche SPoF eines Active/Active Clusters.

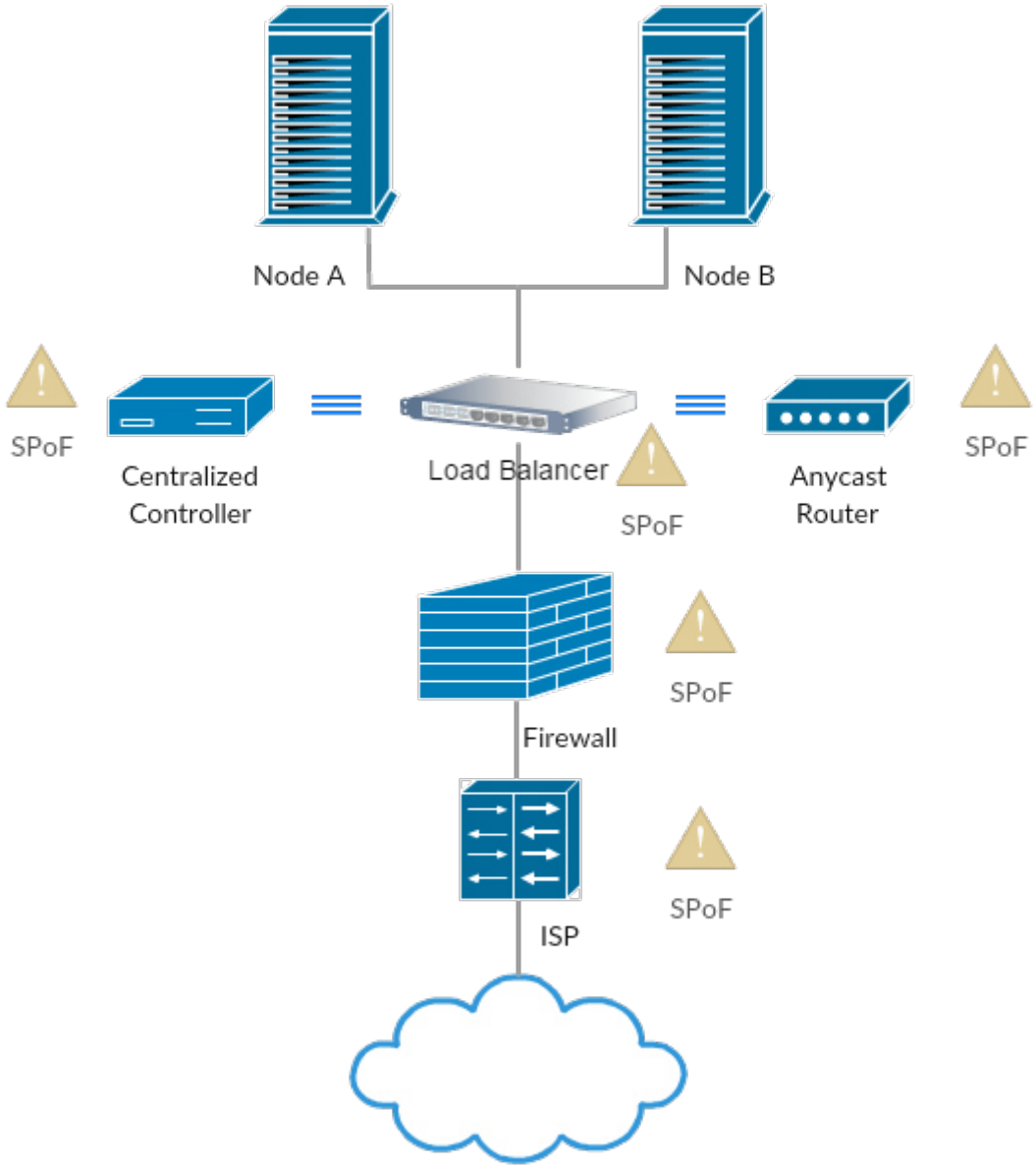

Abbildung 3.4: SPoFs in einem Active/Active Cluster

Es wird deutlich, dass die Planung von Hochverfügbarkeit für Server und Dienste nur einen von vielen Schritten darstellt, die auf dem Weg zu Hochverfügbarkeit gegangen werden müssen. Gerade deshalb soll sich die folgende Konzeption auf das Cluster selbst sowie den Failover eines Nodes beschränken. ISP, Firewall und Loadbalancer werden also nicht weiter thematisiert, dafür liegt der Fokus auf Cluster-Software, OSI-Schicht der Verfügbarkeitsprüfung, den Diensten sowie deren Datenbeständen und Zuständen.

3.3 Herleitung der Erfolgskriterien

Zunächst werden die qualitativen Bedingungen definiert, also solche Kriterien, die bei Prüfung auf Erfolg mit *ja* oder *nein* beantwortet werden. Das Eintreten der erwarteten Ereignisse selbst zu betrachten ist jedoch nicht ausreichend. Zusätzlich müssen maximale Zeitintervalle festgelegt werden, in denen diese stattzufinden haben, damit eine Lösung die Voraussetzungen erfüllt. Die Zeit, in der ein Ausfall erkannt werden muss und soll, um den Gesamtprozess als Erfolg werten zu können, bildet hierfür ein gutes Beispiel. Jener sowie weitere quantitative Werte werden in Kapitel 3.3.2 hergeleitet und festgelegt.

3.3.1 Qualitative Bedingungen

In einem ersten Schritt muss die Cluster-Struktur mitsamt der drei hergeleiteten Nodes durch die Software abbildbar sein. Den logischen zweiten Schritt stellt das Monitoring der Verfügbarkeit des zu betrachtenden Nodes A dar. Als optionales Ziel ist die Möglichkeit zu nennen, das Signal, welches einen Failover auslöst, abgreifen zu können. So soll später in einer produktiven Umgebung eine beliebige Reaktion auf einen Ausfall möglich sein. Die Bandbreite erstreckt sich dabei von Reaktionen eines Loadbalancers, der den inaktiven Node schlicht nicht mehr ansteuert, über einen Failover bis hin zu dem vollständigen Überschwenken des Netzverkehrs auf andere Rechenzentren oder Servercluster.

Im nächsten Schritt muss die Verfügbarkeit des zu überwachenden Dienstes nach einem Failover geprüft werden, der in dieser Phase des Prozesses von Node B angeboten wird. Während die Überwachung des Dienstes selbst obligatorisch ist, wäre ein vollständiges Failover-Monitoring des neuen aktiven Nodes ebenfalls sinnvoll. Daran anschließend stellt sich die Frage, ob die Nutzdaten erhalten bleiben sowie wie viele und welche Zustände dem Dienst in der zweiten Instanz zur Verfügung stehen.

Einen ebenfalls wichtigen Faktor stellt die mögliche Nutzung von selbst definierten Verfügbarkeitschecks dar. Da für das Praxisbeispiel Kamailio die Anforderung gesetzt wurde, die Verfügbarkeit auf Applikationsebene zu überwachen, ist dies unumgänglich. Dabei soll nicht nur auf die Existenz, sondern auch auf die Logik einer Antwort hin geprüft werden. Ein solcher Test muss daher applikationsspezifisch definiert sein. Die erfolgreiche Integration von spezifischen Verfügbarkeitschecks ist somit ein Muss-Kriterium für den Erfolg der gewählten Lösung.

Des Weiteren ist es zwingend notwendig, dass die Cluster-Kommunikation nicht durch den möglicherweise ungeplant wieder aktiv werdenden Node A gestört wird. Handelt es sich bei dem Grund des Failovers um einen temporären Stromausfall oder einen ungeplanten Neustart, wird Node A erneut verfügbar. Dieses Szenario muss bedacht werden. Am sinnvollsten erscheint hier, dass Node B der primäre Node bleibt, bis Node A händisch überprüft und freigegeben worden

ist. Übernähme Node A stets automatisch wieder die Rolle des aktiven Nodes, würde dann jedoch erneut abstürzen und neu hochfahren, wäre keine stabile Verfügbarkeit des Dienstes mehr gegeben. Die hergeleiteten qualitativen Bedingungen für einen erfolgreichen Failover können wie folgt tabellarisch dargestellt werden:

Bedingung	**obl.**	**Messmethode**
Cluster-Struktur abgebildet	ja	Vergleich von Konzept & digitalem Abbild
Überwachung Node A	ja	Cluster Kommunikation & Status
Ausfall registriert	ja	Cluster Kommunikation & Failover-Signal
Failover Signal abgreifbar	nein	Lösungsspezifisch
Dienst auf Node B verfügbar	ja	Verfügbarkeit des Dienstes prüfen
Überwachung Node B	ja	Cluster Kommunikation & Status
Zustände erhalten	ja	Verfügbare Informationen bei Initiierung
Reaktivierung Node A	ja	Dienst bleibt auf Node B
Custom Checks möglich	ja	Prüfung Dokumentation

Tabelle 3.1: Aufstellung der qualitativen Bedingungen

Qualitative Messmethoden

Es soll zunächst händisch auf die korrekte Abbildung der gewünschten Cluster-Struktur geprüft werden. Dabei müssen mindestens zwei, besser aber drei, Nodes in die Failover-Architektur eingebunden sein. Ansatzpunkt für die Erhebung dieser Informationen soll die Überwachung der Kommunikation innerhalb des Clusters im Normalbetrieb, mittels einer Konsole, auf Node A wie B, sein.

Die Prüfung auf korrekte Überwachung des aktiven Node A soll durch obige Methode im Normalbetrieb sowie im Fehlerfall, also durch Deaktivierung von Node A, ebenfalls auf beiden genannten Nodes durchgeführt werden. Analog gilt dies für die Registrierung eines Ausfalls. Die Möglichkeit, das generierte Signal bei einem Ausfall abzufangen, soll durch eine externe Anzeige dessen belegt werden. Realisiert werden soll dies durch Wiedergabe eines Textes auf einer Konsole oder dem Versenden einer E-Mail. Die Verfügbarkeit des Dienstes auf Node B sei durch dessen erneute Nutzung sowie die Cluster-Kommunikation zu prüfen. Insbesondere ist darauf zu achten, ob Clients die erneute Verfügbarkeit automatisch erkennen und somit die Übernahme von Dienst und virtueller IP-Adresse korrekt abläuft.

Die Überwachung der Verfügbarkeit des Node B soll mittels einer Konsole auf einem dritten Node C geprüft werden. Welche und wie viele Zustände erhalten geblieben sind, ist nach erfolgreicher Durchführung vorheriger Schritte durch das Auslesen von bei der Initiierung vorhandenen Informationen innerhalb der Dienste zu prüfen. Dazu muss getestet werden, ob eine

zweite, bereits aktive Instanz die von Node A erhobenen Daten verwenden kann. Bei einer Reaktivierung des Node A soll mittels Cluster-Kommunikation sichergestellt sein, dass Node B den aktiven Status beibehält. Die erfolgreiche Einbindung mindestens einer angepassten Verfügbarkeitsprüfung ist durch Logging-Informationen des Dienstes auf dem aktiven Node sowie der Cluster-Software zu zeigen.

3.3.2 Quantitative Bedingungen

Gerade in Bereichen der Technologie, in denen eine permanente und lückenlose Verfügbarkeit bestimmter Dienste erwartet oder sogar vertraglich garantiert wird, können qualitative Kriterien nicht die einzigen sein, die für eine Prüfung auf Erfolg einer Failover-Infrastruktur herangezogen werden. Vielmehr muss die Frage geklärt sein, welche Werte und Zeiten über- oder unterboten werden müssen. Besonders relevant für die behandelte Thematik sind die durchschnittlich benötigte Zeit bis zur Erkennung eines Ausfalls sowie jene bis der Dienst wieder verfügbar ist. Da Failover-Prozess und Wiedererreichbarkeit von der Erkennung eines Ausfalls abhängen, wird die MTTR der Netzwerkdienste betrachtet. Um die hier konzipierte Lösung möglichst praxisrelevant zu gestalten, muss also eine sinnvolle Größe der MTTR hergeleitet werden.

In der Einleitung wurde bereits darauf eingegangen, wie viel zeitlicher Spielraum dem Anbieter eines Dienstes bleibt. Wer seine Kunden zufriedenstellen möchte oder Verfügbarkeit in bindenden SLA festhält, muss mit kurzen Ausfallzeiten des eigentlichen Dienstes planen. Dabei wird diese Ausfallzeit in der Regel prozentual angegeben. Man spricht also von 8,76 Stunden bei 99,9 Prozent, 52 Minuten bei 99,99 Prozent, 5 Minuten bei 99,999 Prozent und 30 Sekunden bei 99,9999 Prozent garantierter Verfügbarkeit pro Jahr. Um realistische, jedoch nicht überambitionierte Kennzahlen zu entwickeln, wird die sogenannte Carrier Grade von 99,999 Prozent als Basis der folgenden Berechnungen herangezogen.

$$
\begin{aligned}
99,999\% \text{ Verfügbarkeit} &\Rightarrow 0,001\% \text{ Ausfallzeit} \\
0,001\% \text{ (360 Tage)} \mathrel{\widehat{=}} 0,001\% \text{ (8.640 Stunden)} &\mathrel{\widehat{=}} 0,001\% \text{ (518.400 Minuten)} \\
100\% \mathrel{\widehat{=}} \text{518.400 Minuten} \Rightarrow 1\% \mathrel{\widehat{=}} \text{5.184 Minuten} &\Rightarrow 0,001\% \mathrel{\widehat{=}} \text{5,184 Minuten} \\
&\Rightarrow 0,001\% \text{ (360 Tage)} \approx \text{5 Minuten}
\end{aligned}
\tag{3.1}
$$

Im Juni 2013 führte das Marktforschungsunternehmen techconsult GmbH im Auftrag der HP Inc. eine Studie zu IT-Systemausfällen durch. Darin wurden 300 mittelständische Unternehmen

unter anderem darüber befragt, wie viele Ausfälle sie pro Jahr verzeichnen, wie lange diese andauern und wie hoch sie deren Kosten einschätzen. Im Durchschnitt ergaben sich 5 Ausfälle pro Jahr, eine Wiederanlaufzeit 6,5 Stunden sowie Kosten von 24.000€ pro Stunde [tec13]. Eine durchschnittliche MTTR von 6,5 Stunden pro Ausfall steht also 5 Minuten maximaler Ausfallzeit pro Jahr gegenüber. Nimmt man die 5 Ausfälle pro Jahr aus der Studie und addiert einen Puffer von 50 Prozent ergeben sich aufgerundet 8 Ausfälle pro Jahr. Der Puffer wird hinzu addiert, da die erarbeitete Lösung nicht nur ein durchschnittliches Jahr, sondern auch ein Worst-Case Szenario abdecken soll.

Kombiniert man nun die so erhaltenen Daten mit einer garantierten Verfügbarkeit von 99,999 Prozent, ergibt sich folgende Berechnung:

$$5 \text{ Minuten} : 8 \text{ Ausfälle} = 0{,}625 \text{ Minuten} = 37 \text{ Sekunden pro Ausfall} \tag{3.2}$$

Um entsprechende SLA mit 99,999 Prozent garantierter Verfügbarkeit einzuhalten und damit den herzuleitenden Ansatz eines automatisierten Failover als erfolgreich bewerten zu können, darf die MTTR eines Dienstes also maximal 37 Sekunden betragen.

Außerdem sollen mindestens funktional grundlegende Zustände für eine Weiternutzung auf einem Node B erhalten bleiben. Dadurch könnte erreicht werden, dass ein Endgerät in der Telekommunikation die Unterbrechung der Verbindung zu dem Signalisierungsknoten des Anbieters nicht bemerkt und auch keinen neuen Registrierungsprozess inklusive Nutzer- und Standortdaten beginnen muss. Für openHAB ist der Erhalt möglichst vieler Daten und Zustände ebenfalls überaus relevant, da die neue Instanz der Steuerung die zuletzt aktuellen Daten verwenden können soll. Beispiele sind die aktuelle Stufe einer Heizung oder die Lichtstärke einer dämmbaren LED in Prozent.

Die Studie enthält außerdem die Informationen, dass sich über 99 Prozent der befragten Unternehmen bereits im Jahr 2013 mit Hochverfügbarkeit beschäftigt und über 95 Prozent sogar aktive Maßnahmen umgesetzt haben. Zusammengefasst führen die Berechnungen und Festlegungen zu der folgenden Tabelle 3.2.

Bedingung	**obl.**	**Messmethode**
Zeit bis Dienst wieder verfügbar (Max) <= 37 Sek	ja	Laut Monitoring
Zeit bis Dienst wieder verfügbar (Max) <= 37 Sek	ja	Händischer Test
Anzahl Zustände erhalten >= 1	nein	Händische Prüfung / Abfrage

Tabelle 3.2: Aufstellung der quantitativen Bedingungen

Quantitative Messmethoden

Während die qualitativen Werte meist an konkreten Elementen des Clusters zu messen sind, beziehen sich die quantitativen Bedingungen auf den Dienst selbst. Für händische Messungen kann eine Stoppuhr verwendet werden. Mit dieser wird die Zeitspanne von der Deaktivierung des Node A bis zu der Wiederverfügbarkeit eines Dienstes auf Node B gemessen. Bezüglich Kamailio SIP Proxy kann sich dabei zusätzlich auf angemeldete Clients bezogen werden. Relevant sind die Registrierungs-Informationen der Clients innerhalb des serverseitigen Dienstes sowie die Möglichkeit der Initiierung, Beendigung und sonstigen Aktualisierung einer Anruf-Signalisierung. Für openHAB kann die Weboberfläche als Client betrachtet werden, welche auf Node B und innerhalb der gesetzten Zeitspannen, inklusive der zuletzt aktuellen Daten, initiiert und in einem Web-Browser abrufbar sein muss. Da die Performanz eines Clusters aufgrund von Auslastung und Status seiner Teilnehmer variieren kann, werden mehrere Durchläufe vorgegeben, aus deren Ergebnissen die Mittelwerte zu berechnen sind. Überschreitet ein Messwert in mehr als einem Durchgang außerdem den Maximalwert gilt die Anforderung als nicht erfüllt. Für genauere Werte können außerdem Logfiles und Meldungen der Cluster-Software herangezogen werden.

3.4 Arbeitsspeicher und Zustände

In der Diskussion möglicher Ansätze für eine Synchronisation dynamischer Dienste und Zustände muss beachtet werden, dass einige Daten sich ausschließlich im Arbeitsspeicher oder RAM befinden. Viele Dienste und Applikationen halten aktuelle Daten und Zustände im RAM vor, da dieser schnellere Zugriffszeiten bietet als eine externe Datenbank oder Festplattenspeicher. Einen Ansatz zur Synchronisation bildet die regelmäßige Kopie des Speicherstandes. In einer Virtualisierungsumgebung ist es beispielsweise möglich virtuelle Maschinen und in vorliegendem Fall virtuelle Nodes vollständig und inklusive aller vorhandener Zustände zu kopieren. Allerdings beinhaltet diese Vorgehensweise ein grundlegendes Problem: Alle Nodes B bis n halten sich selbst für Node A. Somit können sie nicht als einzigartiger Teilnehmer eines Clusters fungieren und die logische Steuerung des Clusters müsste in einer höheren Abstraktionsebene stattfinden, ohne die Nodes zu involvieren. Außerdem besteht hier die Gefahr der Synchronisierung von defekten Zuständen. Das heißt: Falls der Grund eines Ausfalls des Node A nicht in einem gezogenen Kabel liegt, sondern durch einen Angriff oder korrumpierte Daten hervorgerufen wurde, könnten diese auf alle Klone übertragen werden und zu einer Kettenreaktion führen. Aufwand und Verkehrsaufkommen im Netzwerk durch regelmäßige, vollständige Kopien und oben genannte Aspekte lassen diese Herangehensweise für die hier behandelte Zielsetzung unpassend erscheinen. Außerdem würde der Anwendungsbereich der Lösung hierdurch auf virtuelle Cluster beschränkt. Um die Verbreitung defekter Konfigurationen im Cluster vollständig zu vermeiden,

sollen ausschließlich die verwalteten Nutzdaten automatisch für alle Dienst-Instanzen synchronisiert werden.

3.5 Auswahl einer Clusterstruktur

Unter Beachtung der in den vorherigen Kapiteln genannten Aspekte soll nun eine grundsätzliche Cluster-Struktur festgelegt werden. Aufgrund der Komplexität von Active/Active-Clustern durch Loadbalancer und weiteren, zu beachtenden Techniken sowie der Generierung von zusätzlichen SPoF wird für den Praxisteil ein Active/Passive Cluster festgelegt. Dazu wird die Methode des Hot-Standby gewählt, um die Transferzeit und somit die MTTR des Dienstes möglichst gering zu halten.

Diese Thesis behandelt primär die Verfügbarkeit von dynamischen Diensten und beschränkt daher den strukturellen Kontext sowie weitere SPoF auf die bisherigen Abhandlungen. Diese müssen in einem Produktiv-Cluster also durchaus bedacht werden, spielen für das Testnetzwerk im Praxisteil jedoch keine Rolle. Das Konzept besteht also aus mindestens drei Nodes und einer idealerweise dezentralen Cluster-Management-Software und der optionalen Einbindung in eine vorhandene Monitoring-Umgebung.

Es sei an dieser Stelle angemerkt, dass einzelne Module der im Praxisteil angewandten Methodik auch auf andere Ansätze anwendbar sein können. Die Implementierung der Verfügbarkeitsprüfung auf Applikationsebene beispielsweise ist für alle Cluster-Strukturen relevant. Dies kann theoretisch ebenfalls für die auszuwählende Software-Lösung gelten, da jene gegebenenfalls verschiedene Netzarchitekturen und Failover-Szenarien unterstützt.

3.5.1 Relevanzprüfung aufgeführter Lösungen

Unter Beachtung der gesammelten Informationen ist das Ziel entstanden ein Hochverfügbarkeits-Cluster mit einer Active/Passive Struktur zu konstruieren. Jenes soll ohne zentrale Steuereinheit, also neuen SPoF, auskommen, eine Rollenverteilung sowie eine gegenseitige Überwachung der Nodes ermöglichen. Da somit alle Nodes eine eigene Identität besitzen, scheidet das Konzept der vollständigen Speicherkopie aus. Die aufgeführten Projekte BeeGFS, Ceph, lafs und xtreemfs behandeln zwar ebenfalls eine redundante Struktur, konzentrieren sich jedoch auf die Verwaltung des Festplattenspeichers und bilden somit nur eine kleine Schnittmenge mit der Thematik der Statuserhaltung [IMI; Pro; Divf; Diva]. Da Dienste dynamische und Echtzeitdaten häufig im RAM oder in Datenbanken ablegen, erscheint eine Einarbeitung in diese Software als nicht zielführend. Bei einer Fokussierung auf die Absicherung statischer Daten könnten diese Lösungen jedoch relevant sein.

Nach der im Stand der Technik durchgeführten Recherche scheint die Kombination von Pacemaker und Heartbeat die Anforderungen am besten zu erfüllen. Sollte das Linux Heartbeat komplexen Tests nicht gewachsen sein, kann es durch das leistungsfähigere CoroSync ersetzt werden. Da jedoch beide Module die Einbindung von selbst entwickelten Resource Agents ermöglichen, scheinen alle Bedingungen erfüllt. Es existiert sogar bereits eine Sammlung von vorhandenen Resource Agents für verschiedene Applikationen und Dienste [Comb]. Außerdem funktioniert Pacemaker durch eine Schwarmintelligenz, ist also über alle Nodes des Clusters verteilt und umgeht somit die Bildung eines neuen SPoF. Keepalived und LVS wurden nicht ausgewählt, da sie ihren Fokus auf Loadbalancing und fortgeschrittene Active/Active-Cluster legen. Im Stand der Technik wurde Keepalived als Kombination mehrerer einzelner Funktionalitäten beschrieben, von denen in dieser Arbeit jedoch nur Failover, Überwachung und Statuserhaltung betrachtet werden. HAProxy als Loadbalancer für Webserver trifft ebenfalls nicht den Kern der behandelten Thematik. Die Kommunikation wird über Unicast erfolgen, da kein Loadbalancing zum Einsatz kommt und die Adressierung mittels virtueller IP-Adresse von den Nodes selbst innerhalb des Clusters gesteuert werden soll.

3.5.2 Synchronisierung der Nodes

Wie in Kapitel 2.2.4 angeführt existieren durchaus komplexe Methoden zur teilweisen oder vollständigen Synchronisation von Nodes. Dabei wurde jedoch keine Möglichkeit aufgetan Teile des RAMs abzubilden und zu übertragen. Eine vollständige Kopie eines Node A würde die ausgewählte Cluster-Struktur unmöglich machen, da alle beteiligten Nodes sich selbst für diesen halten würden. Ansätze, wie rsync, wirken auf den ersten Blick sinnvoll und verringern den Aufwand bei Änderungen der Konfigurationsdateien. Es sei nun folgendes Szenario gegeben: Ein Node A schreibt während des Betriebs und aus unbekannten Gründen fehlerhafte Informationen in eine seiner Konfigurationsdateien. Diese Datei wird durch rsync auf einen Node B synchronisiert. Als nächstes führt der kritische Eintrag zu einer Fehlfunktion des Dienstes auf Node A, was die Cluster-Software erkennt und einen Failover einleitet. Bei einem Cold-Standby würde der Dienst auf Node B als Ergebnis gegebenenfalls nicht einmal starten. Bei einem Hot-Standby läuft die Applikation bereits und würde innerhalb absehbarer Zeit das gleiche Verhalten zeigen wie sein Pendant auf Node A. In beiden Fällen ist es wahrscheinlich, dass die Dienst-Instanz den Start während des nächsten Node-Reboots verweigert. Eine gemeinsam verwendete Datenbank wäre von diesem Problem nicht betroffen, da diese nur Nutzdaten des Dienstes vorhält. Der Betrieb des Dienstes selbst wird von Konfigurationsdateien auf den Nodes gesteuert. Die Nutzdaten könnten zwar ebenfalls korrumpiert werden, jedoch fällt dies in den Bereich der redundanten Datenbanken. Wie hierbei Verfügbarkeit garantiert und Daten gesichert werden, ist nicht Teil dieser Arbeit. Daher ist die Synchronisierung der Daten und dynamischen Zustände durch den Ansatz einer gemeinsam verwendeten Datenbank sowie spezifischen Applikations-Einstellungen umzusetzen. Diese sollen erreichen, dass ein Maximum

an Statusdaten (wieder-)verwendbar ausgelagert wird. An diesem Punkt wird deutlich, dass die gewählte Methode in ihrem Funktionsumfang von der Implementierung der zu sichernden Dienste abhängt.

Kamailio beinhaltet mittels des Moduls usrloc die Funktionalität, Userdaten, wie Login und Lokation, in einer Datenbank abzulegen und aktiv zu verwerten [Divd, *usrloc*]. Die Signalisierung von SIP-Calls kann mittels des Dialog Moduls zwar ebenfalls dort abgelegt werden, jedoch ausschließlich zu logging-Zwecken und nicht, um diese wieder zu importieren [Divd, *dialog*]. Es wird Teil der praktischen Ausführung sein die von Kamailio gegebenen Funktionalitäten diesbezüglich genauer zu prüfen. Die Heimautomatisierungslösung OpenHAB ist dagegen in der Lage, alle erhobenen, dynamischen Daten in einer Datenbank zu verwalten und sich selbst mit diesen zu initiieren. Das heißt, dass eine zweite Instanz B mit allen aktuellen, von einer Instanz A erhobenen Daten gestartet werden kann [Dive, *Persistence*].

3.5.3 Das Cluster-Konzept

Das Gesamtkonzept fußt also auf einem Hot-Standby Active/Passive Cluster mit drei Nodes und dem Protokoll Unicast ohne Einsatz eines Loadbalancers. Pacemaker und Heartbeat mit angepassten Verfügbarkeitstests auf Applikationsebene bilden die Steuerungs- sowie Kommunikationsplattform. Hinzu kommen eine gemeinsam verwendete Datenbank und spezifische Einstellungen sowie, falls notwendig, Erweiterungen der betrachteten Dienste Kamailio und openHAB. Im Praxisteil der Thesis soll sich zeigen, in wie weit sich Zustände, die sich bei einer Standardkonfiguration im RAM befinden, für einen Failover synchronisieren lassen und inwiefern dies notwendig ist. Die in Kapitel 2.1.9 und 2.1.10 angeführten Erweiterungen um ein Quorum wird mit zwei erwarteten Votes definiert, womit dem Node C eine entscheidende Rolle bei Split-Brain Situationen zukommt. Zudem kann eine Hochverfügbarkeits-Struktur nur dann effektiv sein, wenn möglichst viele Gründe für einen Ausfall sowie OSI-Schichten abgedeckt werden. Es folgt also die Konzeption der Tests für das 2(-3) Node Active/Passive Cluster.

3.6 Herleitung notwendiger Tests

3.6.1 Ausfall-Szenarien

Die Ausfall-Szenarien decken eine möglichst große Bandbreite ab, um deutlich zu machen, in welch relevanten Bereichen Failover eine Rolle spielt. Dabei sind Teilausfälle ebenso möglich wie der Totalausfall eines ganzen Nodes. Bezüglich der folgenden Auflistung besteht kein Anspruch auf Vollständigkeit.

Gründe für Teilausfälle: Angriffe mittels Daten, Ausfall einer Komponente, Erschütterung, Fehlfunktion eines Treibers, Fehler im Betriebssystem, Fehler innerhalb eines oder mehrerer Dienste selbst, fehlerhaftes Update
Gründe für Totalausfälle: Ausfall der Kühlung, Ausfall einer systemkritischen Komponente, Defekt der Netzteile durch Spannungsspitze, Defekt oder Entfernung der Netzanbindung, (Natur-)Katastrophen, Sachbeschädigung, Stromausfall

3.6.2 Festlegung der Verfügbarkeitstests

Da es sich um die Konzeption einer Ausfallsicherung von dynamischen Netzwerkdiensten handelt, kann an diesem Punkt das OSI-Schichten Modell angewandt werden, um eine logische Struktur der Testplanung zu gewährleisten.

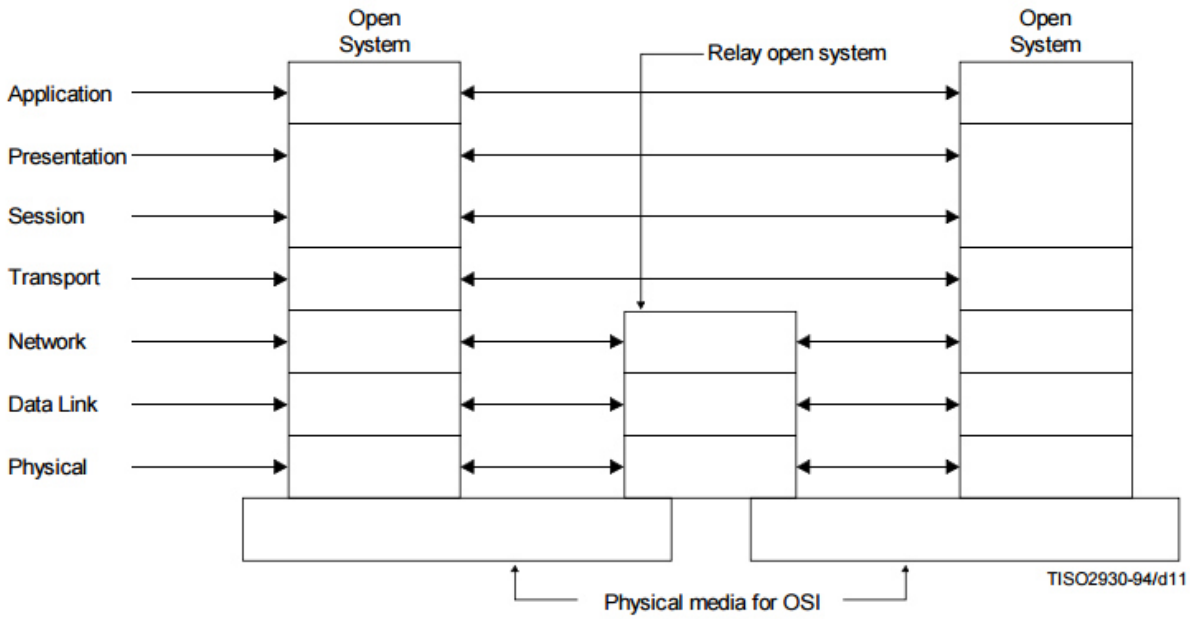

Abbildung 3.5: Unicast, nach [Uni94, *S.29*]

Das OSI-Modell basiert auf Schichten, die wiederum bestimmte Klassen von Protokollen beinhalten und aufeinander aufbauen. Der hierarchische Aufbau ist relevant für die Konzeption der Tests, da ein Dienst, der auf Schicht n angeboten wird, nur genau dann erreichbar und funktional sein kann, wenn auch Schicht n - 1 korrekt arbeitet. Im Umkehrschluss bedeutet dies, dass lediglich auf der höchsten Schicht, der Applikationsschicht, auf Verfügbarkeit geprüft werden muss, um jene für alle Schichten garantieren zu können [Uni94].

Um die Verfügbarkeit einer hochkomplexen Ressource, wie des SIP-Proxys Kamailio oder anderer, dynamischer Netzwerkdienste, gewährleisten zu können, reicht eine reine Prüfung auf Erreichbarkeit nicht aus. Es ist zusätzlich mindestens ein auf Logik basierender Test notwendig. Dass heißt, der überwachte Dienst muss nicht nur innerhalb eines festen Zeitintervalls auf eine Anfrage reagieren, sondern auch die korrekte, zu erwartende Antwort senden. Hierfür sind tiefgehende Kenntnisse des Dienstes an sich sowie die Möglichkeit der Einbindung komplexer, spezifischer Tests notwendig. Ein Test, der eine Anfrage sendet und eine bestimmte Antwort innerhalb eines festgelegten Zeitintervalls zurückerhalten muss, deckt also alle Anforderungen an die Verfügbarkeitsprüfung ab. Es soll abschließend erwähnt werden, dass zusätzliche Tests auf niedrigeren Schichten durchaus sinnvoll sein können. Beispielsweise dann, wenn die Fehlerquelle eines Ausfalls eingegrenzt und möglichst zeitnah gefunden werden soll. Ein Test auf Erreichbarkeit sowie Logik auf der obersten Schicht garantiert Verfügbarkeit. Zusätzliche, verschiedenartige Tests ermöglichen Art und Auftreten eines Fehlers einzugrenzen. Für das in dieser Arbeit angestrebte Cluster-Verhalten reicht ein Test auf Applikationsebene aus.

3.6.3 Anzahl, Intervalle und Grenzen

Zur tabellarischen Aufstellung sowie Auswertung der Messwerte wird ein entsprechendes Programm benötigt. Es wird sich hierbei auf Microsoft Excel festgelegt, wobei auch freie Programme diese Funktionalitäten mitbringen sollten. Die eigentliche Erhebung jener Werte erfolgt händisch und durch das Auslesen von Log-Dateien. Daher erscheinen weitere Tools nicht notwendig. Während die händische Prüfung ungenau ausfallen kann, sollten Logdateien der Cluster-Software sowie der Dienste und deren Clients präziseren Aufschluss über die benötigten Zeiträume geben. In beiden Fällen spielen die konfigurierten Intervalle der Abfragen sowie deren Anzahl bis zu einer Initiierung des Failovers eine große Rolle. Unter Beachtung der maximalen MTTR von 37 Sekunden werden das Sendungs-Intervall der Anfragen auf 2 Sekunden und das Timeout bis eine Anfrage gescheitert ist auf 12 Sekunden festgelegt. Somit wird kein nennenswerter Netzverkehr erzeugt und ein Failover wird bei allen Ausfallarten nach 12 Sekunden Nicht-Verfügbarkeit eingeleitet, welche wiederum einen Zeitpuffer für die Beantwortung der Anfrage unter Last bieten.

In welchem Maße die dynamischen Zustände der betrachteten Dienste beibehalten werden, muss händisch anhand der registrierten Clients, des openHAB Webinterfaces sowie einer während des Failovers aktiven Anrufsignalisierung geprüft werden. Teilerfolge wären die Erhaltung der Verbindungskontexte, erhobener openHAB-Daten und der Registrierungsdaten des Session Initiation Protokolls. Idealziele sind die Aufrechterhaltung eines Telefonats während eines Kamailio Failovers, die korrekte Abhandlung der BYE-Nachrichten des SIP-Protokolls trotz ausgetauschter Proxy-Instanz und die Möglichkeit erneut einen Anruf initiieren, halten oder

weiterleiten zu können. Kombiniert sollten diese Werte und Ziele eine umfassende Abwägung des Erfolges der gewählten Lösung ermöglichen.

3.7 Das finale Modell

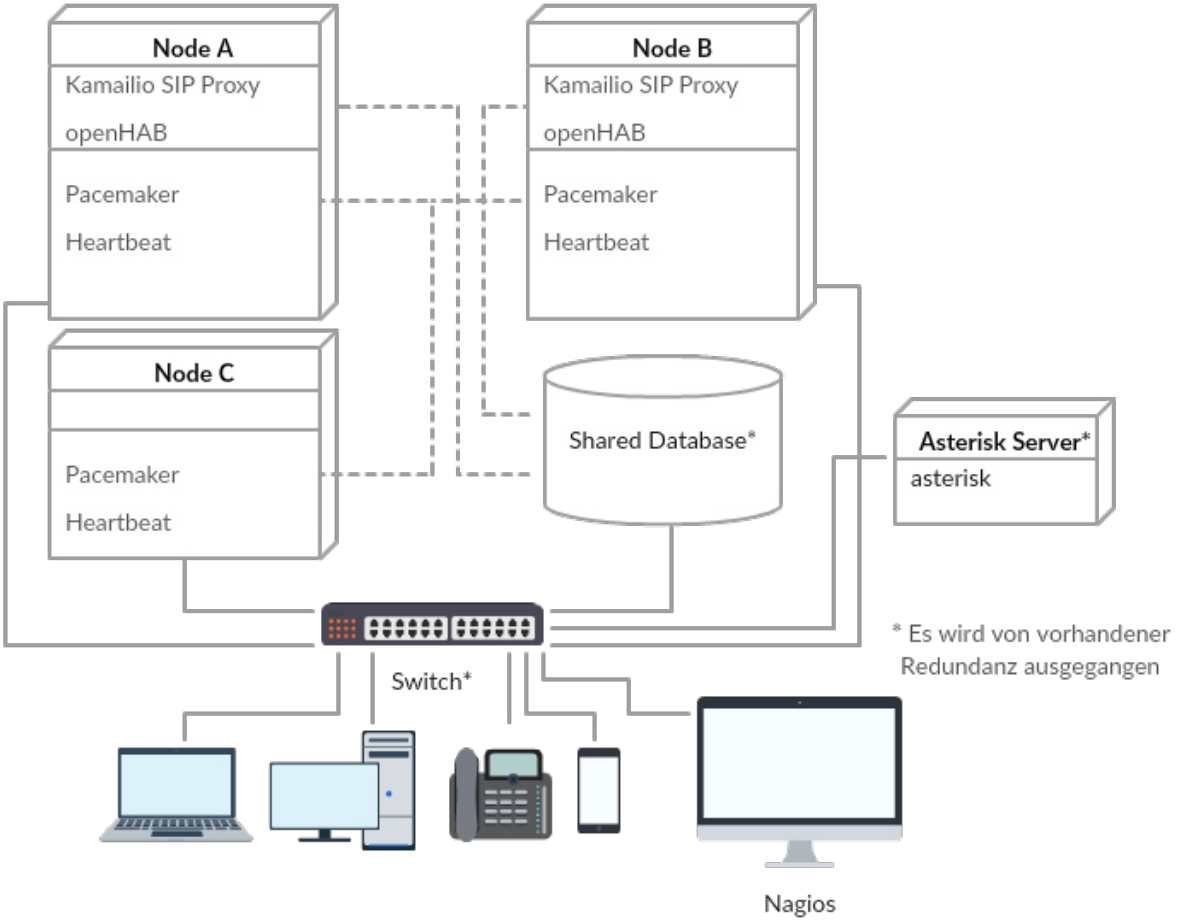

Abbildung 3.6: Konzeptueller Netzplan

Es wird davon ausgegangen, dass die Anbindung der Nodes sowie der Asterisk-Server und die gemeinsam verwendete Datenbank bereits abgesichert und entsprechend redundant zur Verfügung gestellt werden. Durch die Aufteilung von Signalisierung und Echtzeitdatenstrom wird die Netzstruktur der produktiven Cluster der finocom AG im wesentlichen übernommen. Das erhöht den Nutzen der finalen Lösung für das Unternehmen, welches Kamailio den Großteil der SIP-Kommunikation zuteilt, Asterisk aus Abrechnungs- und Loggingbründen jedoch in jedem Gespräch als Zwischenstation der Endstellen einfügt. Die Clients bestehen aus Softphones

sowie der Weboberfläche von openHAB, um verschiedene Testszenarien sowie eine möglichst realistische Auswertung der Ergebnisse zu ermöglichen.

3.8 Abschätzung der zu erwartenden Ergebnisse

Grundlegende Bedingungen für den Erfolg des Gesamtkonzepts, wie die Abbildung der gewünschten Netzstruktur innerhalb der Cluster-Software oder die Überwachung der Nodes an sich, scheinen der Literatur nach nicht überambitioniert. Dazu sollte laut den Quellen zu Pacemaker und Heartbeat die Integration einer logischen, dienstspezifischen Verfügbarkeitsprüfung auf Applikationsebene in Form von Resource Agents realisierbar sein [Divb; Sch12]. Wie komplex die nötigen Veränderungen am Code des Kamailio SIP Proxy sind und, ob solche bei openHAB ebenfalls notwendig sein werden, bildet einen relevanten Teil dieser Arbeit. Mit dem Prozess des Failovers selbst bewegt sich das Projekt bis zu diesem Punkt auf dem Gebiet eines automatisierten Failovers statischer Dienste.

Im Voraus ist schwierig einzuschätzen, in welcher Form das Failover-Signal erzeugt wird und wie dieses abgefangen und extern nutzbar gemacht werden kann. Jenes wird durch die gewählte Cluster-Software implementiert und die Erarbeitung dieses Punktes somit zum einem Part des praktischen Teils. Es folgt die gewünschte, möglichst umfassende Erhaltung der Zustände des jeweiligen Dienstes. Da gewähltes Modell auf externe Replikation der Daten verzichtet und ein Großteil gerade der dynamischen, sich in kurzen Intervallen verändernden Informationen im RAM liegen, gilt es hierbei eine effektive Lösung zu entwickeln. Überaus relevant für dieses Failover-Konzept ist außerdem, inwieweit die Dienste das Auslagern solcher Daten erlauben und jene bei der Initiierung wieder importieren können.

Idealerweise können alle relevanten Zustände und Informationen der betrachteten Dienste nicht nur erhalten bleiben, sondern die von den Diensten angebotenen Leistungen werden während und nach einem Failover nicht oder für den Nutzer kaum merkbar beeinträchtigt. Dazu kommen Aussagen zu den aktuellen Fähigkeiten von Pacemaker, Heartbeat und den verwendeten Modulen sowie das Hervorheben von Ansatzpunkten für folgende Arbeiten unter Verwendung anderer Techniken und Modelle oder einer Weiterentwicklung des in dieser Thesis gewählten Ansatzes.

4 Anwendung des Konzepts

4.1 Allgemeine Struktur

4.1.1 Benötigte Systeme

Aus dem theoretischen Modell in Abbildung 3.6 werden an dieser Stelle das Netzwerk sowie dessen Mitglieder abgeleitet. Realisiert wird das Cluster durch einen Virtualisierungsserver und einen separaten Gateway und Router. Die Nodes A, B und C sowie die gemeinsam verwendete Datenbank existieren demnach in Form virtueller Maschinen. Zusätzlich wird eine weitere VM als Asterisk-Server eingerichtet. Diese steuert in Verbindung mit Kamailio den Echtzeitdatenstrom der Telefonie, womit die logische Struktur dem Aufbau des produktiven Clusters der finocom AG nachempfunden ist. Als Virtualisierungsserver und -software kommen ein Dell Lattitude E7450 mit Intel Core i7-5600U 2,60 GHz Prozessor, 8 Gigabyte DDR3 RAM, einer 256 Gigabyte SSD sowie die freie VirtualBox von Oracle in Version 5.0.10 zum Einsatz. Alle genannten VM, also die Nodes des Clusters, basieren auf dem Betriebssystem Debian 7.9 in einer Minimalinstallation mittels der entsprechenden Net-ISO für AMD64 Systeme [Debb]. Node A und B erhalten jeweils 1024 MB, Node C und die Datenbank jeweils 512 MB RAM, da sie weniger Dienste betreiben müssen. Das Netzwerk wird definiert durch die Netzadresse 10.10.64.0, Netzmaske 255.255.192.0 sowie den Gateway 10.10.64.1 in Form einer pf-Sense und bietet mit 16382 möglichen Teilnehmern ausreichend Raum für Teststellungen.

Node A und B sowie die Datenbank werden wie geplant umgesetzt. Der in der Konzeption eingeführte Node C erhält zusätzlich den Resource Agent ClusterMonitor sowie ein Bash-Script, das von diesem aufgerufen wird. Durch diese Kombination überwacht sich die Pacemaker CIB selbst auf Fehler sowie weitere Meldungen und bietet die Ausführung eines beliebigen Linux-Quellcodes an, was wiederum die gewünschte Schnittstelle zum Cluster-Manager darstellt. Ausgesuchte Überwachungsfunktionen des Pacemaker, wie Start, Stop oder Monitor, können selbstdefinierte Aktionen auslösen, die nur durch die Fähigkeiten des Linux-Systems begrenzt werden. Nicht die spezifischen Resource Agents selbst rufen das Script auf, sondern ein Monitor-Agent, der alle Aktionen jener spezifischen Agenten überwacht. Ein großes Unternehmen kann so bei Störungen beispielsweise den gesamten Netzverkehr auf ein anderes Cluster oder Rechenzentrum umleiten. In der praktischen Umsetzung innerhalb dieser Arbeit sendet

das Bash-Script pcmkFailureNotifier.sh mittels des Message Transfer Agents Exim4 E-Mails an den Cluster-Administrator. Dieser Vorgang findet an der Quelle der Cluster-Steuerung selbst statt und benötigt keine Dritt-Dienste, wie beispielsweise Nagios. Der Agent sowie zugehörige Dateien sind auf allen Nodes konfiguriert. Somit ist auch diese Überwachungsfunktion ausfallsicher ausgelegt und in das eigentliche Cluster integriert. Es sei an dieser Stelle angemerkt, dass das Versenden von E-Mails den Beweis für die Abfangbarkeit des Failover-Signals darstellen soll und externes Monitoring nicht ersetzt. Es ist wichtig, produktive Cluster zusätzlich mittels Anfragen zu überwachen, da selbst ausgelöste Nachrichten bei einer Nicht-Verfügbarkeit ausbleiben können.

Netzplan

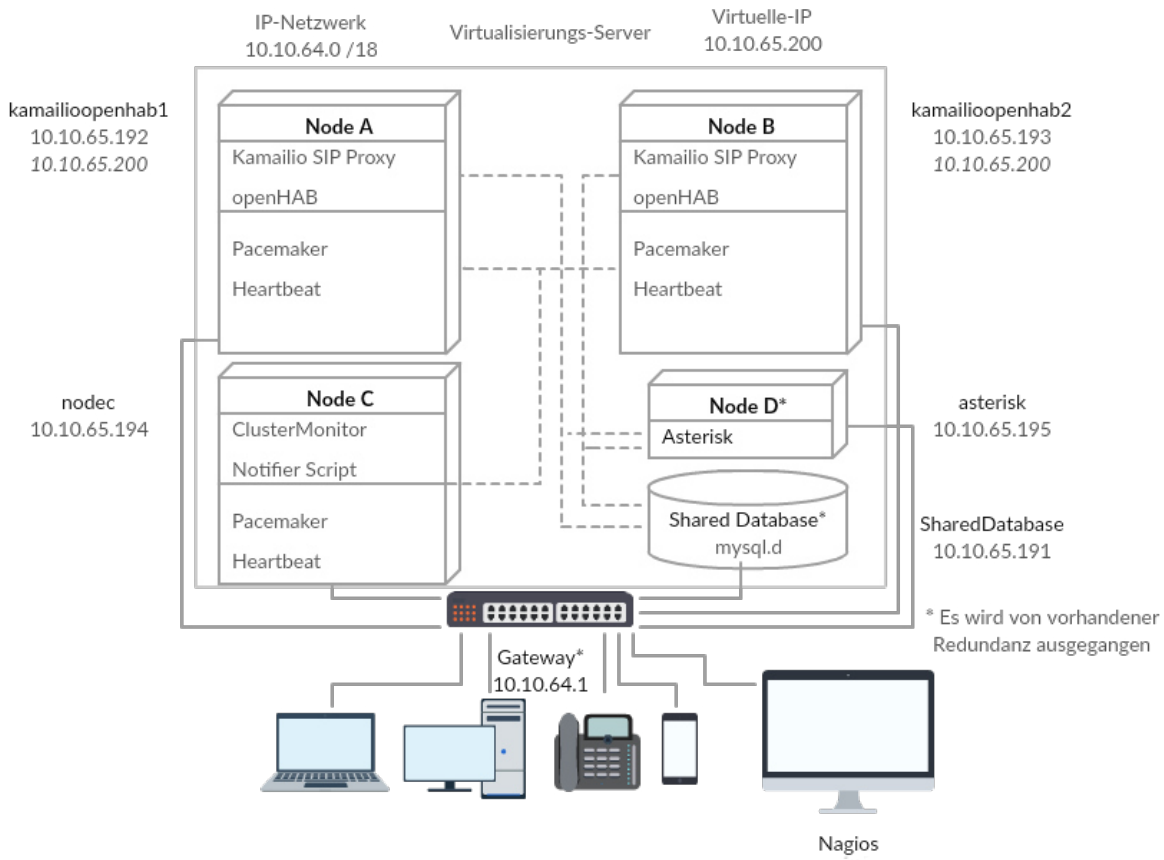

Abbildung 4.1: Praktischer Netzplan

Wie das Modell zeigt, werden die beteiligten Nodes der Teststellung in Form von virtuellen Maschinen auf demselben Server existieren. Dieser Umstand ist in einer produktiven Umgebung unbedingt zu vermeiden, da hierdurch einige essentielle Ergebnisse der Arbeit ad absur-

dum geführt werden würden. Fiele der Virtualisierungsserver aus, wäre die gesamte Failover-Struktur hinfällig. Im realen Umfeld handelt es sich bei den Nodes idealerweise nicht nur um unabhängige, sondern ebenfalls um möglichst unterschiedlich lokalisierte Systeme.

4.1.2 Debian

Mit dem Betriebssystem Debian Wheezy 7.9 wurde sich für eine Linux Distribution entschieden, auf der alle benötigten Dienste laufen können und für die das größte Know-How innerhalb der finocom AG besteht. Außerdem ist Debian in Unternehmen wie wissenschaftlichen Einrichtungen weit verbreitet [Debc] und wird stetig weiterentwickelt. Eben diese Entwicklung kann jedoch auch zu negativen Veränderungen führen. In der Version 8 unterstützt die Distribution Pacemaker nicht nativ, da das Datum für eine Einreichung der Software von den Entwicklern nicht eingehalten wurde [Deba]. Im Dezember 2015 scheint also die stabile und ausgereifte Version 7.9 die beste Wahl zu sein. Dabei wurde eine Minimalinstallation mittels der entsprechenden Net-ISO für AMD64 Systeme verwendet [Debb] und während der Installation die Module SSH Server und Standard System Utilities ausgewählt. Die vollständigen Konfigurationsdateien, wie Interfaces oder resolv.conf sind im Anhang zu finden.

4.1.3 Heartbeat

Als Unterbau der Cluster-Steuerung kommt Heartbeat in Version 3.0.5-3 zum Einsatz. Dieses kann mittels den Befehlen aptitude oder apt-get über die Paketverwaltung bezogen werden und installiert zusätzlich alle benötigten Abhängigkeiten. Auch das Pacemaker-Paket wird durch diese Methode bereits installiert. Im Vergleich mit Corosync wurde Heartbeat für das Projekt ausgewählt, weil seitens der finocom AG auf diese Software verwiesen worden war, sie durch nur zwei Dateien konfiguriert werden kann und dabei scheinbar alle Anforderungen erfüllt. Umso wichtiger ist jedoch der Inhalt der Dateien authkeys und ha.cf, da jene das Fundament des späteren Clusters bilden. Beide Dateien müssen händisch erstellt und mittels chmod 600 mit den vorgegebenen Zugriffsrechten versehen werden.

Die folgende Abbildung zeigt den Inhalt der Datei authkeys, in der Verschlüsselung und Passwort der Cluster-Kommunikation festgelegt werden.

```
1:      auth 2
2:      2 sha1 cluster
```

Quelltext 4.1: authkeys

Es folgt die Festlegung des Logfiles, der Prüfintervalle und der am Cluster beteiligten Nodes in der Datei ha.cf. Letzteres ist notwendig, da der Parameter autojoin auf none gesetzt wird,

um das unerwünschte automatische Beitreten von Nodes in das Cluster zu unterbinden. Der Asterisk-Server selbst ist nicht Teil dieser Arbeit und wird daher nicht mit in das Cluster aufgenommen. Das Prüfungsintervall mittels der Pings wird auf 2 Sekunden und die Zeit, die vergehen darf, bis ein Node als offline deklariert wird, auf 12 Sekunden festgelegt. Somit sollte der gewünschte Failover 12 bis 14 Sekunden nach Ausfall des aktiven Nodes ausgelöst werden. Der exakte Zeitpunkt kann variieren, da eine Abfrage 0 bis 2 Sekunden nach Ausfall des aktiven Nodes gesendet wird, je nachdem, wie lange sein Vorgänger zurückliegt. Für die generelle Kommunikation wird das Unicast-Protokoll definiert.

```
1:     logfile /etc/heartbeat/ha-log
2:     logfacility local0
3:     keepalive 2
4:     warntime 12
5:     deadtime 12
6:     initdead 60
7:     logfacility local0
8:     ucast eth0 10.10.65.192
9:     ucast eth0 10.10.65.193
10:    ucast eth0 10.10.65.194
11:    node kamailioopenhab1
12:    node kamailioopenhab2
13:    node nodec
14:    autojoin none
15:    auto_failback off
16:    pacemaker yes
```

Quelltext 4.2: ha.cf

4.1.4 Pacemaker

Für die fortgeschrittene Umsetzung der Konzeption kommt Pacemaker in Version 1.1.7 zum Einsatz, die über die Paketverwaltung von Debian 7.9 angeboten wird und somit maximale Kompatibilität und Stabilität gewährleistet. Folgende Abbildung zeigt die Ausgabe des Status-Befehls innerhalb der Pacemaker-Konsole:

```
============
Last updated: Sat Jan  2 20:17:40 2016
Last change: Sat Jan  2 20:17:21 2016 via crmd on nodec
Stack: Heartbeat
Current DC: kamailioopenhab1 (fd898711-4c76-4d00-941c-4528e174533c) - partition with quorum
Version: 1.1.7-ee0730e13d124c3d58f00016c3376a1de5323cff
3 Nodes configured, 2 expected votes
4 Resources configured.
============

Online: [ nodec kamailioopenhab2 kamailioopenhab1 ]

 FailoverIP     (ocf::heartbeat:IPaddr2):       Started kamailioopenhab1
 ClusterMon     (ocf::pacemaker:ClusterMon):    Started nodec
 Openhab        (lsb:openhab):  Started kamailioopenhab1
 Kamailio       (ocf::heartbeat:kamailio):      Started kamailioopenhab1
```

Abbildung 4.2: Pacemaker Status

Im Gegensatz zu Heartbeat wird der Cluster-Manager Pacemaker nicht über einzelne Dateien gesteuert, sondern mittels der Kommandozeile auf einem der teilhabenden Nodes die Cluster Information Base angepasst. Die CIB ist dank des Prinzips der Schwarmintelligenz auf allen beteiligten Nodes dieselbe, solange keine Split-Brain Situation eintritt, welche in Abschnitt 2.1.9 näher erläutert wurde. In der CIB sind die Cluster- oder Service-IP und die zu überwachenden Dienste in Form von Resource Agents definiert. Diese erhalten außerdem Parameter zu Adressen, Intervallen und Zeiten, wie sie teils auch für Heartbeat festgelegt wurden. Den Agenten werden außerdem Locations und Colocations zugewiesen. Durch solche kann sichergestellt werden, dass ein Node für einen oder mehrere Dienste bevorzugt oder gemieden wird und, dass bei der Störung eines Dienstes die gesamte Gruppe einen Failover auf einen Node B erfährt. Die zugehörige Konfiguration der CIB ist im Anhang zu finden.

4.1.5 Zusätzliche Dienste

Um OpenHAB auf den Erhalt von Zuständen während eines Failovers prüfen zu können, wurde zusätzlich das SNMP-Protokoll als Client- und Serverversion installiert. Dadurch kann openHAB die Nodes im Cluster nach ergänzenden Informationen abfragen, diese darstellen und abspeichern. Wie sich im Verlauf des Projekts gezeigt hat, bietet jenes Protokoll ebenfalls die Möglichkeit SNMP-Traps zu senden und das Cluster so aktiv mit einer Monitoring-Umgebung kommunizieren lassen zu können. Während die Befehle snmpget und snmpwalk stets von einem Node E auf einen Node D angewandt werden müssen, um Informationen zu erhalten, kann Node D auf diese Weise Änderungen aktiv an Node E propagieren. Diese Umkehrung gleicht dem Ziel des Observer-Pattern in der Programmiertechnik.

Ein weiterer Dienst ist der Message Transfer Agent Exim4, der es unter anderem dem erstellten Notifier-Script und somit Pacemaker ermöglicht, Nachrichten per E-Mail zu versenden.

Abschließend wurde ein SSH Server und der NTP-Dienst installiert, um die Nodes aus der Ferne steuern und sicherstellen zu können, dass Messwerte, Logdateien sowie Nachrichten die korrekte Uhrzeit enthalten.

Shared Database

Dieser Node simuliert in der praktischen Umsetzung des Konzepts eine redundante Datenbankstruktur auf Basis von MySQL. Er ist nicht in den Cluster-Manager eingebunden und bietet eine MySQL-Datenbank sowie Status-Informationen mittels des SNMP-Protokolls an. Sowohl openHAB als auch Kamailio legen hier ihre persistierbaren Daten ab.

Messverfahren & Tools

Das bereits aufgeführte Bash-Script pcmkFailureNotifier.sh sendet E-Mails mit den exakten Werten zu Zeit, betroffenem Node und auslösender Aktion. Außerdem schreibt es jene Informationen für eine bessere Auswertung in die pcmkFailureReceiver Datei auf Node C. Eine Aktion kann dabei Start, Stop, Status, Monitor oder Restart eines überwachten Dienstes mittels des zugehörigen Resource Agents sein. Somit eignen sich die versendeten Nachrichten ideal zur Feststellung quantitativer Werte zu Ausfallerkennung und Wiederverfügbarkeit. Zusätzlich wird händisch und per Überwachung des Cluster-Monitors crm_mon überprüft, ob ein Dienst tatsächlich und vollständig wieder verfügbar ist und die so ermittelten Werte sich mit denen des Scripts decken. Elementar für beide Messverfahren ist die Synchronisierung der Uhrzeit mittels des NTP-Dienstes sowie idealerweise die Verwendung einer Echtzeitabbildung der Atomuhr.

4.2 Kamailio SIP Proxy

4.2.1 Konfiguration

Der SIP Proxy Kamailio wurde in Version 4.3.3 bezogen und beinhaltet mittels des Moduls usrloc die Funktionalität, Userdaten, wie Login und Lokation, in einer Datenbank abzulegen und aktiv wiederzuverwerten [Divd, *usrloc*]. Die Signalisierung von SIP-Calls kann mittels des Dialog Moduls zwar ebenfalls dort abgelegt werden, jedoch ausschließlich zu logging-Zwecken und nicht, um diese wieder zu importieren [Divd, *dialog*]. Für die Aufrechterhaltung eines Anrufs ist dies ausreichend, da der Echtzeitdatenstrom durch den Asterisk-Server gesteuert wird. Wichtig hierbei ist, dass der neuen Kamailio-Instanz die Lokation der Clients sowie deren SIP-Register Status bekannt sind. Nur dann kann sie die Signalisierung mit einem BYE abschließen und die Funktionalität einer sofortigen, neuen Anruf-Signalisierung anbieten. Um

dies zu gewährleisten, wird der Arbeitsmodus des Kamailio userloc-Moduls auf db_mode 3 gesetzt. Kamailio schreibt also alle produzierten Kontaktdaten sofort in die Datenbank. Hier kann theoretisch auch mit dem performanteren db_mode 2 gearbeitet werden, jedoch besteht dann die Möglichkeit, dass gerade die aktuellsten Daten bei einem Failover verloren gehen können. Dieser cached Daten im RAM und schreibt sie in Intervallen in die Datenbank [Mie]. Da eine Abbildung der Konfiguration des Kamailio SIP Proxy mehrere Seiten einnehmen würde, sei an dieser Stelle für weitere Informationen auf den Anhang verwiesen.

4.2.2 Implementation des Resource Agents

Das Git-Repository für Heartbeat Resource Agents hält einen von der Community erstellten Agenten für Kamailio vor [Coma]. Dieser bietet die Möglichkeit einer Verfügbarkeitsprüfung auf Applikationsebene, indem regelmäßig SIP-OPTIONS Pakete an den Kamailio-Server gesendet und dessen Antworten auf Plausibilität überprüft werden. In konkretem Fall bedeutet dies, dass es sich bei der Antwort um ein SIP-Paket 200/OK handeln muss und somit eine Prüfung auf Verfügbarkeit und Logik stattfindet. Damit Kamailio mit dem erwarteten Paket antwortet, müssen Änderungen an dessen Routing-Logik vorgenommen werden, was ein tieferes Verständnis der Applikation sowie des SIP-Protokolls voraussetzt. Es folgt die für den Resource Agent notwendige Anpassung, die vollständigen Konfigurationsdateien sind im Anhang zu finden.

```
1:    ...
2:      # Per SIP request initial checks
3:      route[REQINIT] {
4:        ...
5:        if (is_method("OPTIONS") && ($ru=~"sip:monitor@.*")) {
6:          sl_send_reply("200","Kamailio is alive");
7:          exit;
8:        }
9:        ...
10:     }
11:   ...
```

Quelltext 4.3: kamailio.cfg Auszug

4.2.3 Messungen & Tools

Um die qualitativen Anforderungen zu prüfen, also Aufrechterhaltung eines Telefonats sowie Wiederverfügbarkeit der Anrufsignalisierung, werden zwei Softphones als Clients an dem Kamailio-Server registriert. Die quantitativen Werte können mittels der Meldungen des Notifier-Scripts sowie dem Pacemaker-Monitor crm_mon überwacht werden.

4.2.4 Durchführung des Failovers

Es folgen die ermittelten Werte aus 15 durchgeführten Testläufen, bei denen jeweils ein Failover von Node A auf Node B provoziert werden sollte. In 5 Fällen wurde der Node vom Strom getrennt. In 5 Fällen wurde der Kamailio-Dienst beendet, mittels Umbenennung der primären Konfigurationsdatei korrumpiert und somit ein Neustart verhindert. Für die letzten 5 Durchläufe wurde der Kamailio Port 5060 blockiert, um die Reaktion des Resource Agents sowie des Clusters zu prüfen, wenn die SIP-OPTIONS Pakete nicht mehr beantwortet werden. Die Arten der Durchläufe werden in gleicher Reihenfolge den Buchstaben A, B und C zugeteilt und decken somit einen Totalausfall, ein geordnetes Beenden des Dienstes und eine Nicht-Verfügbarkeit auf Applikationsschicht ab. Für die Auswertung wurde pro Durchgang der höhere Wert aus händischer und automatischer Messung ausgewählt.

#	Art	Failover	Erkennung in s	Verfügbar in s	Zustände
1	A	ja	12	16	ja
2	A	ja	13	22	ja
3	A	ja	13	17	ja
4	A	ja	13	19	ja
5	A	ja	12	17	ja
6	B	ja	4	9	ja
7	B	ja	10	15	ja
8	B	ja	6	10	ja
9	B	ja	5	8	ja
10	B	ja	4	9	ja
11	C	ja	2	20	ja
12	C	ja	2	20	ja
13	C	ja	2	20	ja
14	C	ja	2	19	ja
15	C	ja	2	20	ja

Tabelle 4.1: Testergebnisse Kamailio

Der Mittelwert bis zu der Erkennung eines Ausfalls der Klasse A beläuft sich auf 12,6 Sekunden, der Mittelwert bis zu der vollständigen Wiederverfügbarkeit auf Node B auf 18,2 Sekunden. Für Klasse B wurden 5,8 und 10,2 Sekunden gemessen und für Klasse C waren es 2 und 19,8 Sekunden. In jedem Fall blieben die ermittelten Werte unterhalb der Maximal-Grenze von 37 Sekunden.

4.2.5 Auswertung

Die Auswertung des Kamailio SIP Proxy ist durchweg positiv. Nicht nur arbeiten die Verfügbarkeitsprüfungen aller Schichten korrekt, es wurde außerdem stets und erfolgreich ein Failover durchgeführt sowie der hergeleitete Maximalwert bis zu einer Wiederverfügbarkeit von 37 Sekunden mit höchstens 20 Sekunden deutlich unterschritten. Zusätzlich wird ein aktiver Anruf vollständig gehalten und bei Beendigung auch nach einem Failover korrekt mittels des SIP-Pakets BYE terminiert. Abschließend ist ebenfalls die Funktionalität der Anrufsignalisierung vollständig gegeben. Da dieser Netzwerkdienst auf höchster OSI-Schicht auf Reaktion und Logik geprüft wird, kann von einem vollständigen Erreichen aller Anforderungen und Ziele gesprochen werden.

Weitere Ergebnisse der Testreihen sind, dass ein aktiver Anruf ohne die Failover-Lösung zwar ebenfalls gehalten wird, jedoch weder eine saubere Beendigung des Anrufs noch Weiterleiten oder Halten möglich sind. Eine Initiierung von Anrufen nach dem Ausfall ist unmöglich. Bei einer Struktur mit der Failover-Lösung werden selbst Befehle, die während des Failovers ausgelöst werden, nach Übernahme durch die zweite Instanz durchgeführt. Die Befehle, wie zum Beispiel Weiterleiten, erhalten also lediglich eine Verzögerung bis zur Ausführung und müssen nicht erneut durch den Anwender vorgenommen werden. Fällt der beteiligte Asterisk-Server aus, gleichen die Auswirkungen denen des Kamailio-Ausfalls, da die SIP-Kommunikation nicht korrekt an den Kamailio SIP Proxy zurück übergeben wird. Ist der Asterisk-Server, wie in der produktiven Umgebung der finocom AG, aktiv als Zwischenstelle in den Sprachkanal eingebunden, würde ein Ausfall ebenfalls zu einer Fehlfunktion des aktiven Anrufs führen.

4.3 openHAB

4.3.1 Konfiguration

Die Aufgaben der Heimautomatisierungssoftware openHAB in Version 1.7.1 bestehen in diesem Szenario aus Beschaffung und Anzeige von Daten. Genauer handelt es sich dabei um Status-Daten, die per SNMP-Protokoll von den Nodes A bis C sowie dem Host der Datenbank abgefragt und dann mittels Items sowie einer Sitemap in Form einer Webseite dargestellt werden. Es folgen Auszüge aus der Darstellung der Informationen sowie der Dateien snmp_queries.items und snmp.sitemap, wobei anzumerken ist, dass die Items der Nodes A und B leichte Unterschiede aufweisen, um den aktiven Node selbst als Localhost ansprechen zu können.

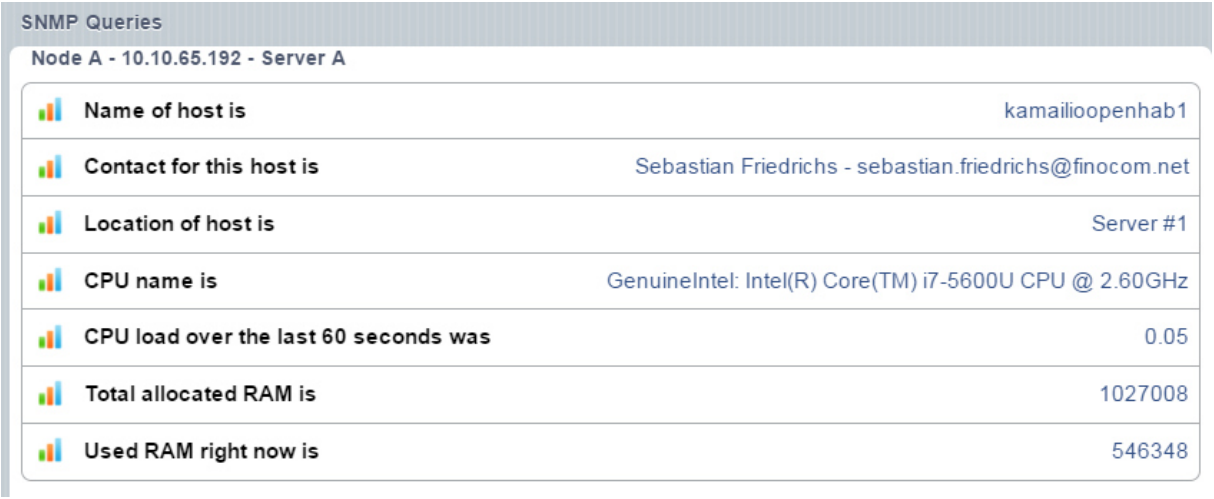

Abbildung 4.3: openHAB Webseite

```
1:     ...
2:     /* Node B on Node A */
3:     String snmpb_1 "Name of host is [%s]" <chart> { snmp="
          <[10.10.65.193:public:1.3.6.1.2.1.1.5.0:180000]" }
4:     ...
5:     String snmpb_7 "Used RAM right now is [%s]" <chart> { snmp
          ="<[10.10.65.193:public:1.3.6.1.4.1.2021.4.6.0:60000]"
          }
6:     ...
```

Quelltext 4.4: snmp_queries.items

```
1:     sitemap snmp label="SNMP Queries" {
2:
3:       Frame label="SNMP Queries" {
4:       ...
5:       Frame label='Node B - 10.10.65.193 - Server B' {
6:       Text item=snmpb_1
7:       ...
8:       Text item=snmpb_7
9:       ...
10:      }
11:    }
```

Quelltext 4.5: snmp.sitemap

Nachdem die vordergründige Funktion des openHAB besprochen wurde, soll nun auf die Sicherung und Erhaltung der Informationen während eines Failovers eingegangen werden. Hierzu wurde die openhab.cfg wie folgt angepasst:

```
1:       ##### General configurations
2:
3:       # The name of the default persistence service to use
4:       persistence:default=mysql
5:       ...
6:       ##### SQL Persistence Service
7:
8:       # the database url like 'jdbc:mysql://<host>:<port>/<
            database>' (without quotes)
9:       mysql:url=jdbc:mysql://10.10.65.191:3306/openHAB
10:
11:      # the database user
12:      mysql:user=openhab
13:
14:      # the database password
15:      mysql:password=openhab
16:
17:      # the reconnection counter
18:      mysql:reconnectCnt=1
19:
20:      # the connection timeout (in seconds)
21:      #mysql:waitTimeout=
```

Quelltext 4.6: openhab.cfg

Zusätzlich muss in einer mysql.persist die sogenannte Strategie der Persistenz definiert werden. Die folgende Datei legt fest, dass alle Items, die openHAB bearbeitet, jeden Tag und bei jeder Änderung in der Datenbank abgelegt werden. Außerdem bewirkt der Parameter restoreOnStartup, dass die Instanz bei einem Start die vorhandenen Werte aus der Datenbank abruft und aktiv verwendet. Somit kann für alle Informationen garantiert werden, dass sie nach einem Failover fortbestehen und genutzt werden können, solange sie von openHAB selbst verwaltet werden. Die Erwartung an das Testergebnis belaufen sich also auf 99 bis 100 Prozent erhaltene Daten und Zustände nach dem Failover. Es folgen die relevanten Zeilen der Persistenz-Datei.

```
1:      // persistence strategies have a name and a definition
           and are referred to in the "Items" section
2:      Strategies {
3:      everyHour : "0 0 * * * ?"
4:      everyDay  : "0 0 0 * * ?"
5:
6:      // if no strategy is specified for an item entry below,
           the default list will be used
7:      default = everyChange, restoreOnStartup
8:      }
9:      ...
10:     Items {
11:       // persist all items once a day and on every change
             and restore them from the db at startup
12:       * : strategy = everyChange, everyDay, restoreOnStartup
13:       ...
14:     }
```

Quelltext 4.7: mysql.persist

4.3.2 Implementation des Resource Agents

Im Dezember 2015 war weder ein OCF-kompatibler Resource Agent für openHAB noch für dessen minimalen Webserver auf Basis von Jetty zu finden. Daher wird für diesen Dienst auf das LSB-kompatible Init-Script zurückgegriffen, das bei einer Installation über die Paketverwaltung des Debian OS enthalten ist. Da es sich hierbei um ein übliches Script zur Dienstverwaltung unter Linux handelt, werden an dieser Stelle keine Code-Auszüge angeführt. Elementar für eine Nutzung in Verbindung mit Heartbeat und Pacemaker sind die Unterstützung sowie korrekte Rückgabeparameter der Funktionen Start, Stop, Status, Monitor und Restart. Der Resource Agent wurde mittels des folgenden Konfigurationseintrags dem Cluster hinzugefügt und durch Zuteilung zu einer Colocation mit den anderen zu überwachenden Diensten verbunden.

```
1:       primitive Openhab lsb:openhab \
2:         meta target-role="Started"
```

Quelltext 4.8: openHAB resource agent

4.3.3 Messungen & Tools

Es ist keine zusätzliche Anpassung notwendig, um die Verfügbarkeit openHABs messen zu können. Für eine Auswertung der Tests werden die E-Mail Benachrichtigungen und Logging Einträge des Cluster-Monitors herangezogen, da sie die genaue Uhrzeit von Nicht- sowie Wiedererreichbarkeit der Dienste protokollieren. Außerdem soll händisch mittels eines Browsers geprüft werden, ob die openHAB Instanz auf Node B korrekt und mit den von Node A erhobenen Daten arbeitet.

4.3.4 Durchführung des Failovers

Es folgen die Ergebnisse aus 10 Testläufen. Davon wurden 5 Failover durch eine Trennung des Node A vom Strom ausgelöst und 5 weitere durch Stoppen des openHAB-Dienstes und dessen Korrumpierung mittels Umbenennung der primären Konfigurationsdatei. Somit werden ein Totalausfall des Nodes A und ein Defekt des Dienstes abgedeckt. Ein Monitoring Klasse C und somit der angebotenen Leistung selbst findet mangels eines passenden Resource Agents nicht statt.

#	Art	Failover	Erkennung in s	Verfügbar in s	Zustände
1	A	ja	12	15	ja
2	A	ja	13	14	ja
3	A	ja	13	17	ja
4	A	ja	13	19	ja
5	A	ja	12	17	ja
6	B	nein	-	-	-
7	B	nein	-	-	-
8	B	nein	-	-	-
9	B	nein	-	-	-
10	B	nein	-	-	-

Tabelle 4.2: Testergebnisse openHAB

Der Mittelwert bis zu der Erkennung eines Ausfalls des Dienstes auf Node A beläuft sich auf 12,6 Sekunden, der Mittelwert bis zu der vollständigen Wiederverfügbarkeit auf Node B auf 16,4 Sekunden. Die Überwachung des Dienstes mittels des zugehörigen Init-Scripts führt dazu, dass ein blockierter Port während des Betriebes nicht bemerkt wird. openHAB startet auch ohne existierende Konfigurationsdatei ohne Fehlermeldung.

4.3.5 Auswertung

Die Testergebnisse der Heimautomatisierungssoftware openHAB fallen mit einer Ausnahme erwartungsgemäß aus. Zur Überwachung der Verfügbarkeit werden IP-Adresse des zugrundeliegenden Nodes sowie der Dienst selbst herangezogen. Die Zeiten bis zu Erkennung eines Ausfalls und der Wiedererreichbarkeit der Applikation bleiben mit durchschnittlich 12,6 und 16,4 Sekunden deutlich unterhalb des geforderten Wertes. Eine fehlende Validierung der Konfigurationsdateien während des Startes verhindert jedoch, dass Korrumpierungen der Einstellungen erkannt werden können. Die Heimautomatisierungssteuerung startet und arbeitet im Normalzustand, während die eigentlich angebotene Leistung nicht verfügbar ist. openHAB selbst bietet die Möglichkeit alle verwalteten Daten in eine MySQL-Datenbank auszulagern und jene während Initialisierung und Betrieb aktiv zu berücksichtigen. Somit konnte eine relative Hochverfügbarkeit mit 100-prozentiger Erhaltung der Informationen und Zustände für diesen Dienst erreicht werden. Ohne eine Integration in die Hochverfügbarkeitsumgebung würden Teil- und Totalausfälle nicht abgefangen werden. Selbst wenn ein Ausfall erkannt und openHAB oder der gesamte Node neugestartet werden würden, könnte eine Schleife von Start bis zu dem erneuten Auftreten der Fehlfunktion entstehen und die von Node A erhobenen Daten ständen Node B nicht zur Verfügung. Diese Problemfälle werden durch die Ergebnisse der Arbeit gelöst. Ideal wäre die Entwicklung eines auf dem Open Cluster Framework basierenden Agenten für openHAB, um maximale Kompatibilität zu Pacemaker zu ermöglichen sowie den Grad der überwachten Verfügbarkeit mittels Logik-Prüfung zu optimieren.

5 Zusammenfassung und Ausblick

Ziel der Arbeit waren Konzeption und Umsetzung einer automatisierten Failover-Lösung, die möglichst viele Informationen und Zustände der überwachten Applikationen erhält. Der Fokus lag auf Netzwerkdiensten, die ihre Funktionalitäten externen Nutzern anbieten und somit nicht nur lokal, sondern auch für Teilnehmer eines Netzwerks permanent erreichbar sein müssen. Dabei wurde seitens der finocom AG ein Linux-basierter Ansatz vorgegeben, da hierfür bereits entsprechendes KnowHow im Unternehmen vorhanden ist. Außerdem sollte für die praktische Umsetzung der auch in den produktiven Clustern eingesetzte Kamailio SIP Proxy behandelt werden. Seitens des Media-Communication Lab der Hochschule Bonn-Rhein-Sieg wurde Open-Source Software präferiert, da diese ebenfalls in weiteren Projekten zum Einsatz kommt und so ein Mehrwert für die Hochschule sichergestellt ist. Hier wurde sich für die praktische Anwendung auf die Heimautomatisierungssteuerung openHAB festgelegt.

Die Anforderungen an das Projekt bestanden in einer Verfügbarkeitsprüfung auf möglichst hoher OSI-Schicht sowie idealerweise einer Plausibilitätsprüfung erhaltener Rückmeldungen. Diese mussten applikationsspezifisch umgesetzt und angepasst werden. Auch sollte das Signal, das einen Failover auslöst, abgreifbar und somit extern nutzbar gemacht werden. Es wurde also eine Schnittstelle angefragt, mit welcher bei Nicht-Verfügbarkeit eines Dienstes zusätzlich oder alternativ zu einem Failover weitere Aktionen definiert werden können. Ideale Ergebnisse bezüglich der Erhaltung dynamischer Zustände des praktischen Beispiels Kamailio SIP Proxy wären die Aufrechterhaltung eines Anrufs während und nach eines Failover-Prozesses und die Möglichkeit, diesen beenden, halten und weiterleiten sowie neue Anrufe initiieren zu können, ohne Änderungen an den Clients vornehmen zu müssen. Da openHAB in der hier verwendeten Konfiguration keine Echtzeitdaten verarbeitet, lag der Fokus darauf, die von einem Node A generierten Informationen bei der Initiierung einer zweiten Instanz auf einem Node B nutzbar zu machen. Zusätzlich sollten alle aktiven Instanzen der beiden betrachteten Netzwerkdienste mit denselben Daten und Zuständen arbeiten. In der Konzeption wurde abschließend ein Wert von 37 Sekunden ermittelt, bis zu dem ein Failover erfolgreich abgeschlossen und alle Dienste wieder verfügbar sein mussten.

Alle oben genannten Ziele und Bedingungen konnten mittels der gewählten Open Source Software bis auf ein Detail erreicht werden. Für openHAB ist im Januar 2016 kein zum Open Cluster Framework kompatibler Resource Agent verfügbar. Daher wird dieser Dienst auf der Applikationsschicht nur auf Existenz und nicht auf Logik seiner Antworten hin überwacht, jedoch kann durch Pings und Prüfung auf Existenz der Prozess-ID Datei, auch PID-File, eine mittlere Verfügbarkeit garantiert werden. Eine sinnvolle Folge dieser Arbeit wäre die Erstellung eines openHAB OCF-Agenten für Heartbeat und Pacemaker. Die zentrale Anlaufstelle hierfür ist auf Github.com zu finden [Coma]. Der Kamailio SIP Proxy wird auf höchster Schicht auf Verfügbarkeit überwacht und die Failover-Lösung garantiert fortlaufende Funktionalität für die relevante SIP-Signalisierung. Bei beiden Netzwerkdiensten bemerken die Clients dank Erhaltung aller relevanten Daten und Zustände nichts von dem eigentlichen Ausfall. Dies gilt für Teil- sowie Totalausfälle der Dienste und des Node A. Die Zielsetzung wurde also erreicht, die praktische Umsetzung kann als erfolgreich angesehen werden, entspricht der Harvard Research Group Klassifizierung AEC-4: Fault Tolerant und bietet mit einer MTTR von maximal 20 Sekunden einen Puffer für mehr als 8 Ausfälle pro Jahr unter Einhaltung einer Verfügbarkeit von 99,999 Prozent. Zudem wurde die MTTR der Netzwerkdienste von der des Node A entkoppelt und es besteht die Möglichkeit der vertikalen Skalierung für das Active/Passive-Cluster. Abschließend ist die STONITH-Technik in vorliegendem Cluster nicht angewandt worden, da mittels des dritten Nodes C das Quorum-Prinzip greift. Entweder es entsteht keine Split-Brain Situation oder der abgeschnittene Node erkennt, dass er in der Unterzahl ist und stört die Kommunikation der anderen Nodes nicht. Außerdem bleiben die Dienste dank der definierten Stickiness stets auf dem Node, auf dem sie gerade aktiv sind. Auch dann, wenn der vorherige wieder aktiv werden sollte.

Für die finocom AG bildet diese Arbeit einen praktischen Nutzen, da die Recherchen, Konfigurationen und Erfahrungen mit dem Clustermanager Pacemaker in die zukünftige Absicherung der produktiven Cluster mit einfließen sollen. Viele auf Linux und openHAB basierende Projekte des Media-Communication Lab könnten durch Nutzung der erlangten Ergebnisse um Ausfallsicherheit erweitert werden, ohne dass die Arbeiten im Kern verändert werden müssten. Der allgemeine Nutzen liegt darin, dass folgende Arbeiten auf den Ergebnissen aufbauen und sich somit der Erweiterung um SharedStorage, der Erforschung fortgeschrittener Synchronisierung der Nodes oder der Programmierung eines OCF-kompatiblen Resource Agents für openHAB widmen könnten.

Des Weiteren konnte ein Überblick über das breite und sich stetig weiterentwickelnde Themenfeld der Hochverfügbarkeit erlangt werden. Diverse kommerzielle und kostenlose Lösungsansätze machen eine eindeutige Festlegung der Anforderungen und Rahmenbedingungen notwendig. Auch muss zwischen Erweiterungen bestehender Systeme und vollständigen, spezialisierten Betriebssystemen unterschieden werden. Die Recherche des kommerziellen Bereichs der Hochverfügbarkeit hat außerdem gezeigt, wie aktuell und brisant das Thema für einige

Unternehmen ist. Zunächst sollte ein SLA eines großen deutschen Anbieters als Basis der Preisberechnungen dienen, was jedoch auf Anfrage in Person der Pressestelle untersagt wurde: „Auf Grund einer internen Entscheidung geben wir [...] keine Zusatzinformationen an die Öffentlichkeit und beteiligen uns auch nicht an Befragungen.“ [Ros15, *Eigene Anfrage*]. Der Anbieter und Global Player nennt also sowohl seine garantierten Verfügbarkeiten als auch deren Preise ausschließlich seinen Kunden. Außerdem gibt die Studie der techconsult GmbH Aufschluss darüber, dass sich über 99 Prozent der befragten Unternehmen bereits im Jahr 2013 mit Hochverfügbarkeit beschäftigt haben.

Literaturverzeichnis

[Ava14] Avaya. *Avaya Aura® Application Enablement (AE) Services High Availability (HA) White Paper*. AEServicesConsolidatedHighAvailabilityWhitePaper.pdf. 2014 (zitiert auf Seite 2).

[BK11] Oliver Beren Kaul. *Skalierbarkeit*. RESTLab-Skalierbarkeit-Oliver-Beren-Kaul-kurz-und-gute.pdf. 2011 (zitiert auf Seite 17).

[Cas] Adam Cassen Alexandre & Fletcher. *LVS NAT + Keepalived HOWTO*. URL: http://www.keepalived.org/LVS-NAT-Keepalived-HOWTO.html (besucht am 23. 12. 2015) (zitiert auf Seite 30).

[Cas02] Alexandre Cassen. *Keepalived for LVS*. KeepalivedUserGuide.pdf. 2002 (zitiert auf Seite 30).

[Cis01] Cisco. *Handbuch Netzwerktechnologien*. München: Markt-und-Technik-Verl., 2001 (zitiert auf Seite 21).

[Col04] Daniel Collins. *Carrier Grade Voice Over IP*. 2. Auflage. Carrier Grade Voice Over IP.pdf. McGraw-Hill, 2004 (zitiert auf Seite 24).

[Coma] Clusterlabs Community. *Github Resource Agents*. URL: https://github.com/ClusterLabs/resource-agents/tree/master/heartbeat (besucht am 27. 12. 2015) (zitiert auf Seiten 54, 63).

[Comb] Heartbeat Community. *resource-agents-heartbeat*. URL: https://github.com/ClusterLabs/resource-agents/tree/master/heartbeat (besucht am 15. 12. 2015) (zitiert auf Seite 42).

[Deba] Debian.org. *Debian HA FAQ*. URL: https://wiki.debian.org/Debian-HA/FAQ (besucht am 27. 12. 2015) (zitiert auf Seite 50).

[Debb] Debian.org. *Mirror Archive*. URL: http://cdimage.debian.org/mirror/cdimage/archive/7.9.0/amd64/iso-cd/ (besucht am 26. 12. 2015) (zitiert auf Seiten 48, 50).

[Debc] Debian.org. *Who is using Debian*. URL: https://www.debian.org/users/ (besucht am 27. 12. 2015) (zitiert auf Seite 50).

[Diva] Diverse. *A fault-tolerant distributed file system for all storage needs*. URL: http://www.xtreemfs.org/ (besucht am 03. 11. 2015) (zitiert auf Seite 41).

[Divb] Diverse. *ClusterLabs Mailing List*. URL: http://clusterlabs.org/wiki/Mailing_lists (besucht am 23. 09. 2015) (zitiert auf Seite 47).

[Divc] Diverse. *Fast incremental file transfer*. URL: https://rsync.samba.org/ (besucht am 03. 11. 2015) (zitiert auf Seite 20).

[Divd] Diverse. *Kamailio Modules - v4.3.x*. URL: http://kamailio.org/docs/modules/stable/ (besucht am 06. 11. 2015) (zitiert auf Seiten 43, 53).

[Dive] Diverse. *openhab Wiki Persistence*. URL: https://github.com/openhab/openhab/wiki/Persistence (besucht am 06. 11. 2015) (zitiert auf Seite 43).

[Divf] Diverse. *The Least-Authority File Store*. URL: https://tahoe-lafs.org/trac/tahoe-lafs (besucht am 03. 11. 2015) (zitiert auf Seite 41).

[Dud] Bibliographisches Institut GmbH: Duden. *Duden.de*. URL: http://www.duden.de/ (besucht am 21. 10. 2015) (zitiert auf Seiten 4, 6, 27).

[Eas06] Perhelion Easyas12c. *Wikipedia Routing Schemata*. 2006. URL: https://de.wikipedia.org/wiki/Anycast (besucht am 18. 11. 2015) (zitiert auf Seiten 21, 22).

[Erb12] Benjamin Erb. "Concurrent Programming for Scalable Web Architectures". Concurrent Programming.pdf. Diploma Thesis. Institute of Distributed Systems, Ulm University, 2012 (zitiert auf Seiten 18, 19).

[Eur15] Host Europe. *Host Europe SLA*. HostEuropeSLA.pdf. 2015 (zitiert auf Seiten 6, 24).

[Gro] Network Working Group. *Request for Comments*. URL: https://tools.ietf.org/rfc/index (besucht am 18. 11. 2015) (zitiert auf Seiten 12, 22).

[hap] haproxy.org. *Description*. URL: http://www.haproxy.org/#desc (besucht am 23. 12. 2015) (zitiert auf Seite 31).

[Hat] Red Hat. *Red Hat Pricing Catalog*. URL: https://www.redhat.com/wapps/store/catalog.html (besucht am 02. 11. 2015) (zitiert auf Seite 26).

[IEE06] IEEE762TF. *IEEE Standard Definitions for Use in Reporting Electric Generating Unit Reliability, Availability, and Productivity*. 762-1 Task Force IEEE762TF-762-2006.pdf. 2006 (zitiert auf Seite 5).

[IET98] The Internet Society: IETF. *rfc2460; Internet Protocol, Version 6 (IPv6) Specification*. 1998. URL: https://tools.ietf.org/html/rfc2460#section-2 (besucht am 29. 10. 2015) (zitiert auf Seite 2).

[IMI] Fraunhofer Institute for Industrial Mathematics ITWM. *The parallel cluster file system*. URL: http://www.beegfs.com/content/ (besucht am 03. 11. 2015) (zitiert auf Seite 41).

[Inf] Bund für Sicherheit in der Informationstechnik. *Cloud Computing Grundlagen*. URL: https://www.bsi.bund.de/DE/Themen/DigitaleGesellschaft/CloudComputing/Grundlagen/Grundlagen_node.html (besucht am 12. 09. 2015) (zitiert auf Seite 1).

[Lie13] Oliver Liebel. *Linux Hochverfügbarkeit: Einsatzszenarien und Praxislösungen*. 2. Auflage. Galileo Computing, 2013 (zitiert auf Seiten 3–5, 8–12, 14–17, 29).

[LIN] LINBIT. *Distributed Replicated Block Device*. URL: http://drbd.linbit.com/ (besucht am 03. 11. 2015) (zitiert auf Seite 20).

[Mie] Daniel-Constantin Mierla. *usrloc Module*. URL: http://kamailio.org/docs/modules/3.1.x/modules_k/usrloc.html#id2995804 (besucht am 27. 12. 2015) (zitiert auf Seite 54).

[Oraa] Oracle. *Oracle Linux Download*. URL: http://www.oracle.com/technetwork/server-storage/linux/downloads/default-150441.html (besucht am 02. 11. 2015) (zitiert auf Seiten 25, 26).

[Orab] Oracle. *Oracle Solaris 11*. URL: http://www.oracle.com/technetwork/server-storage/solaris11/overview/index.html (besucht am 02. 11. 2015) (zitiert auf Seite 25).

[Orac] Oracle. *Oracle Solaris Cluster*. URL: http://www.oracle.com/technetwork/server-storage/solaris-cluster/overview/index.html (besucht am 02. 11. 2015) (zitiert auf Seite 25).

[Pho14] Saleem N. Bhatti & Ditchaphong Phoomikiattisak. *Fast, Secure Failover for IP*. Fast Secure Failover for IP.pdf. 2014 (zitiert auf Seite 2).

[Pre] SAP Presse. *global.sap.com*. URL: http://global.sap.com/germany/newsreader/index.epx?articleId=25011 (besucht am 14. 10. 2015) (zitiert auf Seite 23).

[Pro] Ceph Project. *The Future of Storage*. URL: http://docs.ceph.com/docs/v0.80.5/start/intro/ (besucht am 03. 11. 2015) (zitiert auf Seite 41).

[RH03] Inc. Red Hat. *Delivering High Availability Solutions with Red Hat Cluster Suite*. RHAClusterSuiteWPPDF.pdf. 2003 (zitiert auf Seite 7).

[Ros15] Uwe Rosenbaum. *Telekom Stellungnahme 2*. TelekomStellungnahme2.pdf. 2015 (zitiert auf Seite 64).

[Sch12] Michael Schwartzkopff. *Clusterbau: Hochverfügbarkeit mit Linux*. 3. Auflage. OReilly Verlag GmbH & Co. KG, 2012 (zitiert auf Seiten 3, 6, 9, 10, 26–30, 47).

[Sta04] Matthew Stafford. *Signaling and Switching for Packet Telephony*. 2. Auflage. Artech House Inc, 2004 (zitiert auf Seite 25).

[Sym13] Symantec. *Strategies to Solve and Optimize Management of Multi-tiered Business Services*. solve-management-of-multi-tier-business-apps.pdf. 2013 (zitiert auf Seite 2).

[tec13] techconsult. *techconsult Ausfall Studie*. 2013. URL: http://www.datacenter-insider.de/software-on-premise/rz-tools/articles/407378/ (besucht am 18. 11. 2015) (zitiert auf Seite 39).

[Uni94] International Telecommunications Union. *X.200: Basic Reference Model: The basic mode*. T-REC-X.200-199407.pdf. 1994 (zitiert auf Seiten 8, 44).

[Wen98] Wensong. *What is virtual server*. 1998. URL: http://www.linuxvirtualserver.org/whatis.html (besucht am 23. 12. 2015) (zitiert auf Seite 30).