Bachelor-Thesis

„Konzeption einer Hochverfügbarkeitslösung für LAMP-Systeme mit Open-Source-Software und Standard-Hardware“

eingereicht von

Bernd Michael Helm

geboren am 18.09.1987 in Pirna
Martikelnummer 4715172

Fachhochschule der Wirtschaft Dresden
Studiengang: Angewandte Informatik

Betreuer:	Prof. Dr. Winfried Lachnit Fachhochschule der Wirtschaft Dresden
1. Gutachter:	Prof. Dr. Winfried Lachnit Fachhochschule der Wirtschaft Dresden
2. Gutachter:	Dipl. Ing. Martin Gaber b.i.b. Dresden
Tag der Themenübergabe:	18.03.2010
Tag der Einreichung:	26.07.2010

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

BSI	Bundesamt für Sicherheit in der Informationstechnik
CMS	Content Management System
CPU	Central Processing Unit
CRM	Customer Relationship Management
DNS	Domain Name System
EAN	European Article Number
FTP	File Transfer Protocol
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
IMAP	Internet Message Access Protocol
IP	Internet Protocol
LAN	Local Area Network
POP3	Post Office Protocol 3
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SSL	Secure Socket Layer
SVN	Subversion
TCP	Transmission Control Protocol
TTL	Time To Live
URL	Uniform Resource Locator
WLAN	Wireless Local Area Network

Glossar

Administrator	Person, welche für die Betreuung von IT-Systemen verantwortlich ist
Apache	Quelloffener und weit verbreiteter Webserver
Arbeitsspeicher	Hauptspeicher für Programmdaten und Zwischenergebnisse eines Computers
Backup	Sicherheitskopie von Daten
Betriebssystem	Software, welche den Betrieb eines Computers ermöglicht
Browser	HTTP-Client zum Abrufen und Darstellen von Webseiten
C	Programmiersprache
C++	Weiternetwicklung der Programmiersprache C
Client	Software oder System, welches Serverdienste in Anspruch nehmen kann
Code	Für Menschen lesbare Version eines Programmes in einer Programmiersprache
Courier	Modularer E-Mail-Server für Unix und unixähnliche Betriebssysteme
Daten	Logisch gruppierte Informationseinheiten
Datentyp	Menge von Objekttypen mit bestimmten Operationen
Debian	Populäre Linux-Distribution
Domain	Für Menschen lesbare Kommunikationsadresse für Computersysteme und Dienste
E-Mail-Adresse	Adresse zum Empfangen und Senden von E-Mails
Ethernet	Technik für ein kabelgebundenes Datennetz
Feature	Funktionalität einer Software
Firefox	Quelloffener Webbrowser des Mozilla-Projekts
Freie Software	Quelloffene Software, die für jeden Zweck verwendet werden kann
GPL	Copyleft-Lizenz für freie Software
Host	Computer, der Dienste zur Verfügung stellt
Hosting	Internetdienste bereitstellen
HTTP	Übertragungsprotokoll für Webseiten und Dokumente im Internet
Hypervisor	Zentraler Bestandteil einer Virtualisierungssoftware

Internet Explorer	Webbrowser vom Softwarehersteller Microsoft für dessen Betriebssystem Windows
IP-Adresse	Eindeutige Identifikationsnummer für Systeme in einem Netzwerk
Kernel	Zentraler Bestandteil eines Betriebssystems
Kompilieren	Übersetzen vom Quellcode zu ausführbarem Maschinencode
Linux	Freies, quelloffenes und unixähnliches Betriebssystem
Master	Hierarchische Stellung über einem Slave
MySQL	Quelloffenes, relationales Datenbankverwaltungssystem
Open Source	Lizenzen für quelloffene Software
PHP	Skriptsprache mit C-ähnlicher Syntax
PHP-Skript	Ein in PHP geschriebenes Programm
Portabel	Auf verschiedene Systeme übertragbar
Postfix	E-Mail-Server für Unix und Unixähnliche Betriebsysteme
Proprietäre Software	Unfreie Software
Protokoll	Vereinbarung zum Austausch von Daten zwischen Computern oder Prozessen
Proxy	Vermittler, welcher sowohl als Client, als auch als Server agiert
Prozess	Ein gerade ausgeführtes Computerprogramm
Quellcode	Für Menschen lesbare, in einer Programmiersprache geschriebener Text eines Computerprogramms
Query	Datenbank-Abfrage
Routing	Festlegung für Wege von Datenströmen
Scriptsprache	Meist interpretierte Programmiersprachen, die vor allem für kleine, überschaubare Programmieraufgaben gedacht sind
Server	Soft- oder Hardware, die anderen Dienste anbietet
Shell-Script	Für Kommandozeileninterpreter erstelltes Skript
Slave	Hierarchische Stellung unter einem Master
Subnetz	Ein Teil eines Netzwerkes
Swap-Speicher	Festplattenspeicher, welcher an Stelle von Arbeitsspeicher verwendet wird
Traffic	Datenaufkommen bei Computernetzwerken

Webapplikation	Anwendung zur dynamischen Generierung von Webseiten
Webserver	HTTP-Server, welcher Dokumente an Clients überträgt
Webshop	Webapplikation mit Warenkorbfunktion über die Waren im Internet gekauft werden können
While-Schleife	Syntaxelement einer Programmiersprache, welche bedingte, schleifenartige Ausführung von Codeteilen ermöglicht
Windows	Proprietäres Betriebssystem von Microsoft
World Wide Web	Über das Internet abrufbare, mit einander verknüpfte Dokumente

1 Einführung

1.1 Einleitung

Kleine und mittelständige Unternehmen setzen oft eigene Server für Kundenprojekte ein. Im Vergleich zu Hosting-Angeboten, bei denen keine Kontrolle über den Server möglich ist, bietet dies vor allem Flexibilität, eventuell aber auch Vorteile bei Leistung, Datenschutz und Kosten. Auf der anderen Seite liegt die Administration dieser Server in der Verantwortung der Unternehmen. Bei einer Hosting-Lösung eines namenhaften Anbieters sorgt dieser für Backups und Ausfallsicherheit, bei einem eigenen Server gehört dies jedoch zu den Aufgaben des Administrators.

Die Firma Helm & Walter IT-Solutions, welche im Weiteren als „das Unternehmen“ bezeichnet wird, setzt z.Z. zwei Server ein, um Kundenprojekte wie z.B. Webshops zu hosten und zu betreuen. Zwar kommt eine Backup-Strategie zum Einsatz, doch im Falle eines kompletten Serverausfalls wäre mit einer Ausfallszeit von vier bis zehn Stunden zu rechnen. Dies kostet Zeit und Geld, da ein neuer Server bestellt und installiert, alle Daten aus dem Backup extrahiert und die Kundenprojekte wieder eingerichtet werden müssen. Noch schlimmer ist es für den Kunden, wenn durch den Ausfall des Webshops Umsatz verloren geht. Schon eine Nichterreichbarkeit von wenigen Stunden kann zu Kundenunzufriedenheit und bei längerem Ausfall zu einem Zerbrechen der Kundenbeziehung führen.

Der Serverausfall muss vom Verantwortlichen möglichst sofort behandelt werden, um die Webshops so schnell wie möglich wieder online zu bringen. Dies kann schwierig werden, wenn der Server außerhalb der Geschäftszeiten ausfällt, besonders wenn der zuständige Serveradministrator gerade im Urlaub oder krank ist.

Ziel dieser Bachelor-Thesis ist es, für das Unternehmen eine später produktiv einsetzbare, Linux-basierende Open-Source-Hochverfügbarkeitslösung für Webapplikationen zu entwerfen. Die Konzeption und letztendlich auch die Implementierung einer Testumgebung dieser Lösung wird unter Verwendung eines osCommerce-Webshops durchgeführt. Prinzipiell soll die Architektur die Ausführung anderer PHP/MySQL-basierender Webanwendungen erlauben.

1.2 Aufbau und Struktur der Arbeit

Zu Beginn der Arbeit werden die Grundlagen betrachtet, welche für das Verständnis der Problemanalyse und der darauffolgenden Kapitel notwendig sind. Anschließend wird der bisherige Zustand der Hosting-Architektur des Unternehmens festgehalten und analysiert. Außerdem werden wichtige Daten und die Anforderungen an eine neue, hochverfügbare Hosting-Architektur definiert. Die Entwicklung der Lösung beginnt anschließend mit der Analyse und dem Vergleich von möglichen Konzepten zum Erreichen der definierten Anforderungen. Nach der Entscheidung für ein konkretes Konzept werden die Vorraussetzungen für dieses genauer untersucht und beschrieben. In den letzten Kapiteln dieser Arbeit wird das Konzept in einer Testumgebung realisiert. Anschließend wird es mit einem Kundenprojekt getestet, um die Anwendbarkeit für das Unternehmen festzustellen. Die Testergebnisse werden ausgewertet und miteinander verglichen. Im letzten Kapitel, dem Fazit, werden die erarbeiteten Zwischenergebnisse zu einem Gesamtkonzept zusammengefasst. Dieses kann dann vom Unternehmen realisiert werden.

2 Grundlagen

Sehr viele Webanwendungen sind in PHP geschrieben und setzen MySQL als Datenbank ein. Dies umfasst viele populäre Shop-, Foren- und CMS-Systeme. Diese Architektur ist betriebssystemunabhängig und gut portabel. Die Kommunikation mit dem Client wird durch das zustandslose HTTP-Protokoll abgewickelt, was sich theoretisch gut dafür eignet, diese Anfragen auf mehrere Systeme zu verteilen. Dieses Grundlagenkapitel definiert die wichtigsten Begriffe, welche für das weitere Verständnis nötig sind. Da sich diese Thesis auch zu einem großen Teil mit MySQL befasst, umschließen die nötigen Grundlagen auch ein prinzipielles Verständnis von SQL.

2.1 Was bedeutet LAMP?

Erstmalig erwähnt wurde diese Abkürzung 1998 von Michael Kunze im c't Magazin[1]. LAMP ist ein Akronym für dem kombinierten Einsatz des Betriebssystems **L**inux, dem Webserver **A**pache, des Datenbankservers **M**ySQL und der Skriptsprache **P**HP. Dabei handelt es sich um eine Software-Infrastruktur für die Entwicklung und Ausführung von Webapplikationen, welche im Wesentlichen aus den genannten Teilkomponenten besteht. Der Apache ist mit über 50% Marktanteil[2] der populärste Webserver der Welt. Die MySQL-Datenbank steht ihrerseits an dritter Stelle der am meisten eingesetzten Datenbank-Systeme.[3] Auch die Programmiersprache PHP befindet sich an 4. Stelle der TIOBE-Programmiersprachenübersicht[4] und ist somit ebenfalls weit verbreitet. Alle drei Technologien haben gemeinsam, dass sie freie Software und damit kostenlos sind. Die meisten Linux-Distributionen stellen die genannten Komponenten zur Verfügung, meist schon für ein Zusammenspiel kompiliert, vorkonfiguriert und dokumentiert[5], so dass schnell ein funktionsfähiges System eingerichtet werden kann. Auch gibt es auf der Website *apachefriends.org* eine erweiterte LAMP-Distribution unter der Bezeichnung XAMPP für Windows und Linux.[6]

2.2 Was bedeutet Hochverfügbarkeit?

Hochverfügbarkeit (engl. *high availability*, abgekürzt HA) bedeutet, dass ein System trotz Ausfall einer oder mehrerer seiner Komponenten verfügbar bleibt, ohne dass dazu menschliches Eingreifen erforderlich ist.[7] Dies bezieht sich nicht nur auf IT-Systeme, sondern lässt sich auf jedes kritische System übertragen. Die Verfügbarkeit

1 Kunze, Michael, c't 12/98, Seite 230 - LAMP: Freeware Web Publishing System with Database Support (1998), http://web.archive.org/web/20071212203835/http://www.heise.de/ct/english/98/12/230/
2 Netcraft Ltd, April 2010 Web Server Survey (2010), http://news.netcraft.com/archives/2010/04/15/april_2010_web_server_survey.html
3 Oracle Corporation, Market Share (2010), http://www.mysql.com/why-mysql/marketshare/
4 TIOBE Software BV, TIOBE Programming Community Index for July 2010 (2010), http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
5 HowtoForge, How To Set Up A Ubuntu/Debian LAMP Server, http://www.howtoforge.com/ubuntu_debian_lamp_server
6 Seidler, Kai, XAMPP (2010) http://www.apachefriends.org/de/xampp.html
7 Dupke, Kai, Der Weg zu Hochverfügbarkeitslösungen (2001), http://research.intelego.net/heise/ix/2001/04/089/art.htm

eines Systems wird meist prozentual angegeben und bezieht sich auf eine gewisse Zeitbasis wie ein Monat oder ein Jahr.

Klasse	Verfügbarkeit	Ausfallzeit pro Monat	Ausfallzeit pro Jahr
2	99,00%	7,2 h	87,66 h
3	99,90%	43,2 min	8,77 h
4	99,99%	4,43 min	52,59 min
5	99,999%	0,43 min	5,26 min

Tabelle 1: Verfügbarkeit und Ausfallzeiten pro Monat bzw. Jahr.

Ein Monat entspricht 30,43 Tagen und ein Jahr 365,25 Tagen

Quelle: Nach „O`Reilly Clusterbau mit Linux-HA Version 2“ 1. Auflage 2008 Seite 2

Die Formel für die Berechnung der Verfügbarkeit eines Systems lautet wie folgt:

$$Verfügbarkeit = \frac{Gesamtzeit - Gesamtausfallzeit}{Gesamtzeit}$$

Der Begriff „Hochverfügbarkeit“ ist an keine Verfügbarkeitsklasse oder Prozentzahl gebunden, sondern steht oft dafür, dass Maßnahmen ergriffen wurden, um die Verfügbarkeit gegenüber eines Einzelsystems zu verbessern.

2.3 Verbundene Systeme

Wenn Kunden beispielsweise einen Online-Shop und Webhosting kaufen, dann erwarten diese, dass der Shop 24 Stunden pro Tag und 7 Tage pro Woche, also *immer* verfügbar ist. Hinter dem Online-Shop steht eine Kette von technischen Systemen, die dafür sorgen, dass dieser über das Internet erreichbar ist. Diese beginnt bei der eingesetzten Shop-Software, der Server-Software (Webserver, Datenbank) sowie dem Server-Betriebssystem. Sie geht weiter über die Server-Hardware, die Stromversorgung durch das Rechenzentrum, das Netzwerk (Kabel, Switches) bis hin zur Internetverbindung des Rechenzentrums. Fällt eines dieser, unter anderem in Abbildung 1 gezeigten, Systeme aus, ist der Shop nicht mehr erreichbar. Sind die Verfügbarkeiten der verketteten und voneinander abhängigen Teilsysteme bekannt, so lassen sich diese multiplizieren, um die Verfügbarkeit des Gesamtsystems zu errechnen.

$$Gesamtverfügbarkeit = \prod Teilverfügbarkeiten$$

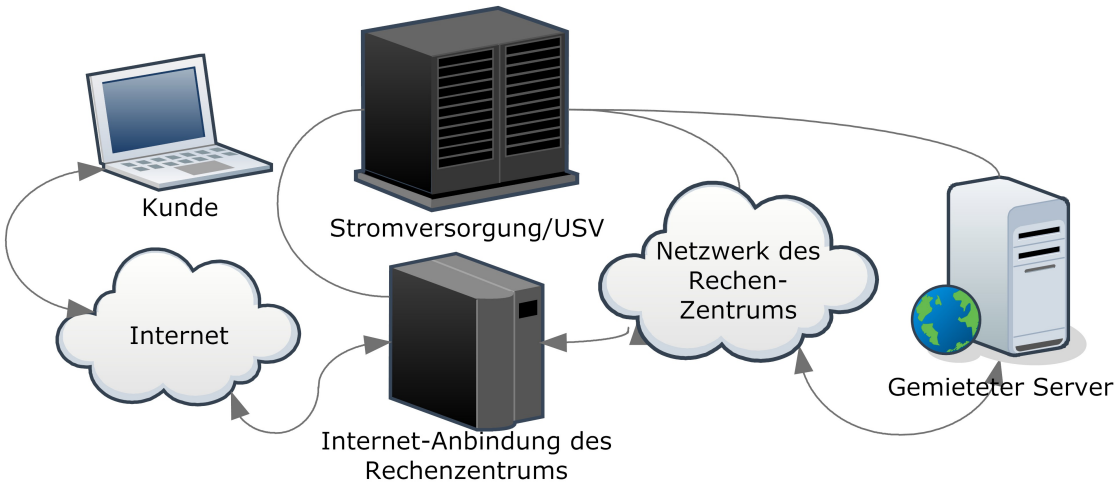

Abbildung 1: Sehr stark vereinfachte Darstellung der Umgebung eines Servers in einem Rechenzentrum und der im Text erwähnten Komponenten

Quelle: eigene Abbildung

Wenn beispielsweise die Verfügbarkeit der Infrastruktur des Rechenzentrums 99,90% im Jahr beträgt und die Verfügbarkeit des Servers 99,90%, so errechnet sich die Gesamtverfügbarkeit wie folgt:

$$0{,}999 * 0{,}999 = 0{,}998001$$

Die Gesamtverfügbarkeit für dieses Beispiel beträgt also 99,80%.

Soll die Verfügbarkeit eines Servers maximiert werden, indem im Notfall ein zweiter seine Aufgaben übernehmen kann, so handelt es sich um zwei parallel geschaltene, redundante Systeme. Das Gesamtsystem ist nur dann nicht mehr verfügbar, wenn beide Server gleichzeitig ausfallen.

$$Gesamtverfügbarkeit = 1 - \prod (1 - Teilverfügbarkeit_x)$$ [8]

$$1-(1-0{,}999)*(1-0{,}999) = 1-(1-0{,}999)^2 = 0{,}999999$$

Wenn beide Server jeweils eine Verfügbarkeit von 99,90% besitzen, so beträgt die Gesamtverfügbarkeit beider parallel gekoppelter Systeme 99,9999%

8 Vgl. Schwartzkopff, Michael, Clusterbau mit Linux-HA Version 2 (2008) Seite 6

2.4 Was bedeutet Lastverteilung?

In der Informatik bedeutet Lastverteilung (engl. *load balancing*, abgekürzt LB), dass eine Menge an Anfragen auf mehrere parallel arbeitende Systeme verteilt werden, um so die Arbeitsgeschwindigkeit gegenüber eines einzelnen Systems zu erhöhen. Zum Beispiel werden bei heute üblichen Mehrkern-Prozessoren die auf dem Computer laufenden Prozesse auf die verfügbaren Prozessorkerne verteilt.

Ein weiteres Beispiel für Lastverteilung sind zwei oder mehr gleich konfigurierte Web-Server, die HTTP-Anfragen entgegennehmen und bearbeiten. Die Anfragen können auf die vorhandenen Server verteilt werden, so dass beide Server im Idealfall nahezu gleich ausgelastet sind. Sofern diese nur statische Inhalte verteilen, müssen sie nicht einmal etwas von der Existenz der anderen Server wissen. Wenn sie aber komplexere Systeme bereitstellen, bei denen die Anfragen Daten verändern, muss ein Austausch der Änderungen zwischen allen beteiligten Servern stattfinden, um die Gleichheit der Systeme zu bewahren.

2.5 Computer-Cluster

Cluster ist der englische Begriff für „Haufen“, „Schwarm“ oder „Gruppe“ und bezeichnet in der Informatik einen Zusammenschluss von Computern, die gemeinsam arbeiten.[9] Die beteiligten Computer sind miteinander verbunden, so dass Informationen ausgetauscht werden. Ein Cluster kann sowohl in Hardware, in Software als auch in Kombination von beidem implementiert werden. Im weiteren Verlauf der Thesis ist mit dem Begriff Cluster immer eine Software-Lösung gemeint, welche auf Standard-Hardware ausgeführt wird. Die Systeme, die den Cluster bilden werden in der Cluster-Terminologie als Knoten bezeichnet. Manchmal sind mit Knoten auch Betriebssystem-Prozesse einer Software-Implementierung gemeint.

2.5.1 Failover / Aktiv/Passiv-Cluster

Failover, engl. für „Ausfallsicherung“ beschreibt in der Informationstechnik eine Technologie, mit der Daten und Dienste hochverfügbar gehalten werden können. Im Zusammenhang mit Serverdiensten spricht man oft von einem Failover- oder

9 Vgl. Schwartzkopff, Michael, Clusterbau mit Linux-HA Version 2 (2008) Seite 17

Aktiv/Passiv-Cluster.[10] Bei einem Aktiv/Passiv-Cluster existiert ein aktives Primärsystem, welches alle Anfragen bearbeitet. Außerdem gibt es ein oder mehrere passive Standby-Systeme, welche jederzeit die Aufgaben des aktiven Systems vollständig übernehmen können, falls dieses unerwartet ausfällt. Das bedeutet, dass alle zum Betrieb notwendigen Informationen auf allen Knoten vorhanden und synchron sein müssen, da sonst bei einem Ausfall des Primärsystems nicht übertragene Informationen verloren gehen. Auch muss dafür gesorgt werden, dass nach dem Wechsel des Systems alle Anfragen auf das nun aktive Standby-System geleitet werden.

2.5.2 Aktiv/Aktiv-Cluster

Wie bei dem Aktiv/Passiv-Cluster gibt es auch hier mindestens 2 Systeme, welche beide dieselben Aufgaben übernehmen können. Anders als beim Aktiv/Passiv-Cluster nehmen hier aber alle Knoten Anfragen entgegen und bearbeiten diese. Aus diesem Grund ist die Leistungsfähigkeit eines Aktiv/Aktiv-Clusters tendenziell größer als bei einem Aktiv/Passiv-Cluster mit derselben Anzahl an Systemen, insofern es keine Ausfälle gibt. Fällt ein Knoten aus, so übernehmen die Verbleibenden dessen Aufgaben.[11] Problematisch ist, dass bei einem Aktiv/Aktiv-Cluster durch den Ausfall eines Knotens die Leistungsfähigkeit des gesamten Clusters sinkt. So sind hierbei zwar die gewünschten Dienste noch verfügbar, aber ggf. merklich langsamer oder gar unbenutzbar. Bei der Planung eines Aktiv/Aktiv-Clusters muss also beachtet werden, dass der Cluster auch nach Ausfall eines Knotens noch genügend Kapazitäten besitzt um die gewünschten Aufgaben zu erfüllen. Außerdem müssen die Kapazitäten ausreichen um den eventuell sehr aufwendigen Neusynchronisierungsprozess auszuführen.

10 Vgl. Bundesamt für Sicherheit in der Informationstechnik, HV-Kompendium V1.2 - Datenbanken (2009), Seite 32, https://www.bsi.bund.de/cae/servlet/contentblob/483628/publicationFile/30947/9_Datenbanken_pdf.pdf

11 Vgl. Bundesamt für Sicherheit in der Informationstechnik, HV-Kompendium V1.2 - Datenbanken (2009), Seite 30, https://www.bsi.bund.de/cae/servlet/contentblob/483628/publicationFile/30947/9_Datenbanken_pdf.pdf

2.6 Netzwerklatenz

Die Datenübertragung in einem Computernetzwerk erfolgt über Pakete, welche neben den eigentlichen Nutzdaten auch Informationen des verwendeten Netzwerkprotokolls enthalten, die für die Zustellung des Paketes wichtig sind. Die Übertragung eines Paketes von einem Programm zum Anderen erfolgt nicht sofort. Die Daten werden von dem Programm an das Betriebssystem übergeben, welches sie in entsprechende Pakete verpackt und dann an die Netzwerkschnittstelle weitergibt. Anschließend werden die Daten über das Netzwerk übertragen. Je nach Übertragungsmedium und Entfernung des Ziels, sowie der Leistungsfähigkeit und Anzahl der dazwischen liegenden Netzwerkkomponenten, wie Router und Switches, gelangt das Paket letztendlich an die Netzwerkschnittstelle des entfernten Rechners. Dort wird es wieder ausgepackt und an das entfernte Programm übergeben. Sind mehr als diese zwei Rechner im Netzwerk, kann es passieren, das anderer Netzwerkverkehr die Übertragung behindert, so dass deren Dauer zusätzlich ansteigt. Die Netzwerklatenz kann mittels ICMP, beispielsweise über den Ping-Befehl, bestimmt werden. Dabei wird die Zeit eines Paketes zum entfernten Rechner und zurück gemessen. Die typische Latenz in einem Ethernet-Netzwerk beträgt etwa 0,2 bis 0,4 Millisekunden.

2.7 Virtuelle Maschinen

Bei einer virtuellen Maschine handelt es sich um eine virtuelle Laufzeitumgebung. Innerhalb dieser Laufzeitumgebung können je nach Beschaffenheit einzelne Prozesse oder ganze Betriebssysteme ausgeführt werden. Eine virtuelle Maschine für Betriebssysteme simuliert die Hardware eines Computers und kann so auch als Abstraktionsschicht zwischen Hardware und Betriebssystem und der darin ausgeführten Software betrachtet werden. Diese Abstraktion ermöglicht es, mehrere Betriebssysteme auf einem physikalischen System auszuführen und auch die virtuellen Maschinen zwischen verschiedenen physikalischen Systemen zu bewegen. Dabei muss das virtualisierte Betriebssystem nicht modifiziert werden. Alle Referenzen auf virtuelle Maschinen innerhalb dieser Thesis meinen die Virtualisierung von Betriebssystemen. Die physikalische Hardware und das darauf laufende Betriebssystem wird als Host-System bezeichnet, die virtuellen Maschinen dagegen als Gäste.

2.8 Cron-Jobs

Der Cron-Deamon ist ein Unix-Dienst ähnlich dem Task Planer unter Windows, welcher standardmäßig bei den meisten Linux-, BSD- und Mac OS X-Distributionen vorinstalliert ist. Die Aufgabe des Cron-Deamons ist es, zeitlich gesteuert wiederkehrende Befehle und Programme auszuführen[12]. Diese Aufgaben werden als Cron-Jobs bezeichnet. Neben Wartungsaufgaben des Betriebssystems werden sie auch oft in Verbindung mit Webapplikationen eingesetzt. Beispielsweise können sie bei einem Webshop aktuelle Preise und Lieferzeiten vom Großhändler direkt in den Shop importieren und diese wieder an eBay oder Amazon exportieren. Wenn in dieser Thesis von Cron-Jobs die Rede ist, dann sind diese Importe/Exporte und ähnliche Hintergrundaufgaben gemeint.

2.9 Über MySQL

Der MySQL-Datenbank-Server ist ein relationales Datenbankverwaltungssystem, welches unter anderem für Unix/Linux, Windows und Mac OS X verfügbar ist[13]. MySQL ist als Open-Source Community-Version unter GPL und als proprietäre Enterprise-Variante verfügbar. Die Aufgabe von Datenverwaltungssystemen wie MySQL ist es, Informationen widerspruchsfrei zu speichern und angefragte Teilmengen in bedarfsgerechter Form bereitzustellen. Bis 2008 wurde MySQL von dem schwedischen Unternehmen MySQL AB entwickelt, welches dann für 1 Milliarde US-Dollar von Sun Microsystems übernommen wurde.[14] Im Januar 2010 wurde Sun und somit auch MySQL vom Datenbank-Spezialisten Oracle für 7,4 Milliarden US-Dollar gekauft.[15]

MySQL ist nach eigenen Angaben[16] die populärste Open-Source-Datenbank der Welt und außerdem Teil des Paketbestandes der meisten Linux-Distributionen, wodurch sich die Installation einfach gestaltet. Bekannte Unternehmen und Projekte wie

12 The Open Group Base, Specifications Issue 7, Crontab (Datei) (2008), http://www.opengroup.org/onlinepubs/9699919799/utilities/crontab.html
13 Oracle Corporation, MySQL Referenzhandbuch, 2.4.2. Operating Systems Supported by MySQL Community Server (2010), http://dev.mysql.com/doc/refman/5.0/en/which-os.html
14 Heise Online, Sun kauft MySQL AB für eine Milliarde US-Dollar (2008) http://www.heise.de/newsticker/meldung/Sun-kauft-MySQL-AB-fuer-eine-Milliarde-US-Dollar-Update-179176.html
15 Golem.de, Oracle kauft Sun (2009), http://www.golem.de/0904/66578.html
16 Oracle Corporation, Über MySQL (2010) http://www.mysql.de/about/

Google, Wikipedia, Yahoo, T-Systems, Siemens, Symantec, Flickr, Adobe, Facebook, Twitter, Youtube und viele weitere setzen die MySQL-Datenbank bzw. MySQL-Cluster produktiv ein.[17] Als Gründe dafür werden neben der Anpassbarkeit durch Open-Source-Software und die Kostenersparnis auch die Performance und Funktionalität von MySQL genannt.

MySQL unterstützt offiziell mehrere *Storage-Engines*, oder auch Tabellentypen, was dem Entwickler einer Datenbankanwendung die Möglichkeit gibt, zu wählen wie und wo die Daten gespeichert werden. Die beiden bekanntesten Tabellentypen sind MyISAM und InnoDB. MySQL ist in C/C++ programmiert und beherrscht seit Version 5.0 weitestgehend den SQL-Standard.[18]

2.10 SQL-Grundlagen

SQL steht für „*Structured Query Language*“, was wörtlich übersetzt „Strukturierte Abfrage Sprache“ bedeutet. Dabei handelt es sich um eine Datenbanksprache, welche zur Abfrage und Manipulation von Datensätzen und zur Definition von Tabellen verwendet wird. SQL wird von ISO standarisiert, wobei die aktuelle Revision „SQL:2008“[19] als auch ältere Revisionen des Standards nicht kostenlos verfügbar sind. Die meisten relationalen Datenbankverwaltungssysteme unterstützen SQL, wobei die Implementierungen der einzelnen Systeme derart von einander abweichen, dass die SQL-Abfragen zum Teil nicht ohne Modifikationen auf ein anderes Datenbanksystem übertragbar sind.

Viele Webapplikationen benötigen eine SQL-Datenbank um zu funktionieren. Bei dem in dieser Thesis betrachteten osCommerce-Webshop werden Kundendaten, Bestellungen, Artikeldaten, Kategorien und Ähnliches in einer MySQL-Datenbank gespeichert. In den folgenden Unterkapiteln werden die wichtigsten SQL-Befehle und der grundlegende Aufbau einer SQL-Datenbank beschrieben. Die Beschreibung orientiert sich dabei an dem SQL-Dialekt von MySQL. Die einzelnen Befehle und deren

17 Oracle Corporation, MySQL-Customers (2010), http://www.mysql.com/customers/ und unterseiten.
18 Oracle Corporation, MySQL-Referenzhandbuch, 1.8.5. MySQL Differences from Standard SQL (2010)
http://dev.mysql.com/doc/refman/5.0/en/differences-from-ansi.html
19 International Organization for Standardization, ISO/IEC 9075-1:2008 (2010), http://www.iso.org/iso/catalogue_detail.htm?csnumber=45498

Syntax werden in einer vereinfachten Form beschrieben. Ziel dieser Einführung ist ein grundlegendes Verständnis der SQL-Befehle, weswegen auf vollständige Syntax-Schemen mit allen optionalen oder alternativen Elementen verzichtet wird. Die vollständige Syntax der SQL-Befehle kann auf den MySQL-Handbuchseiten nachgelesen werden. Die SQL-Syntax-Elemente der Befehle werden, wie es auch in der Dokumentation von MySQL üblich ist, in Großbuchstaben innerhalb des Textes angegeben.

2.10.1 Strukturierung der Daten

Aus logischer Sicht werden die Daten innerhalb von MySQL in Tabellen und Datenbanken organisiert. Ein MySQL-Server kann mehrere Datenbanken verwalten und diese können mehrere Tabellen beinhalten. Die Tabellen innerhalb einer Datenbank gehören meist zu einer Anwendung. Datenbanken dienen zur logischen Gruppierung von Tabellen und werden zur Vergabe von Nutzerrechten verwendet. So kann ein Nutzer beispielsweise das Recht erhalten, innerhalb einer Datenbank neue Tabellen zu erstellen, zu ändern oder zu entfernen.

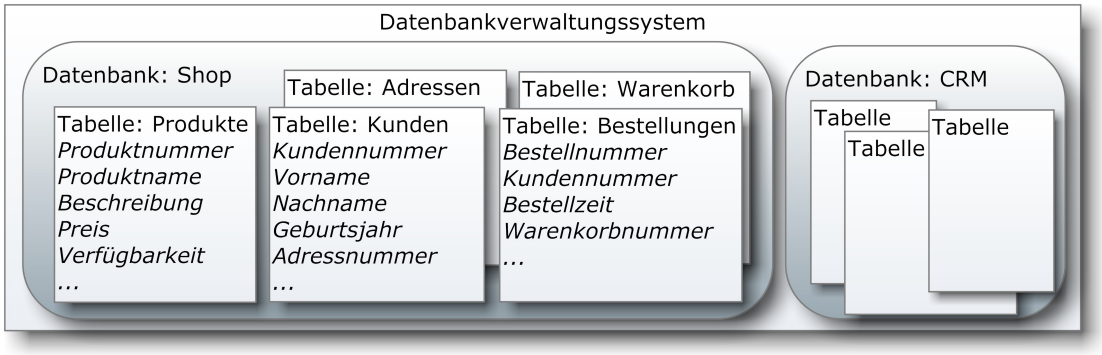

Abbildung 2: Beispiel für Datenbanken und Tabellen eines Datenbankverwaltungssystems

Quelle: eigene Abbildung

Datenbanktabellen besitzen einen Namen, unter dem sie angesprochen werden können und sind in Zeilen und Spalten gegliedert, in denen die Daten strukturiert gespeichert werden. Tabellen ähnlich wie die der Datenbank „Shop“ aus Abbildung 2 werden von dem osCommerce Webshop verwendet. Die Daten innerhalb einer Spalte sind immer vom selben Datentyp, also in Art und Beschaffenheit gleich. Wird beispielsweise eine Spalte einer Tabelle als ganzzahlig definiert, so können in dieser

keine Texte gespeichert werden.[20] Wie schnell eine Suche in einer Spalte erfolgt, wieviel Speicherplatz die Daten darin belegen und welche Operationen auf diese Spalte ausgeführt werden können, hängt von deren Datentyp ab.

2.10.2 Lesende Befehle: SELECT

Der SELECT-Befehl wird verwendet um Informationen aus einer oder mehreren Tabellen abzufragen. Die einfachste Variante eines SELECT-Befehls zum Auslesen von Daten aus einer Tabelle lautet *„SELECT * FROM `Tabellen-Name`“*. Dies gibt alle Spalten und Zeilen der Tabelle als Ergebnis zurück. Soll nur ein Teil der Datensätze zurückgegeben werden, muss die gewünschte Ergebnismenge mit der WHERE-Klausel definiert werden, welche auch bei dem Befehl UPDATE zum Einsatz kommt. Mit Hilfe des SELECT-Befehls lassen sich viele Problemstellungen allein über die Datenbank lösen, welche sonst aufwendig zu programmieren wären. Dabei wird anders als bei den meisten Programmiersprachen nicht der Weg zur Lösung spezifiziert, sondern nur wie das gewünschte Ergebnis aussehen soll. Wie die Lösung genau ermittelt wird, ist Sache des Datenbanksystems. Dies kann von dem Programmierer nur indirekt durch die Anpassung der Fragestellung verändert werden. SELECT selbst verändert keine Datensätze und ist somit nur eine lesende, nicht datenverändernde Operation.

Soll eine Abfrage Informationen liefern, die sich nur unter Verwendung von mehr als einer Datenbanktabelle ermitteln lassen, so müssen die beteiligten Tabellen mit einem sogenannten „JOIN“ (engl. für Vereinigung) zusammengeführt werden. Es gibt mehrere Arten von Joins, wobei der „Equi-Join“ (engl. für Gleichheits-Vereinigung) am meisten Verwendung findet. Das Ergebnis eines einfachen Joins von zwei Tabellen ist das kartesische Produkt oder auch Kreuzprodukt aller Zeilen der ersten, mit allen Zeilen der zweiten Tabelle. In der Regel sind nur die Zeilen dieses Kreuzproduktes gesucht, die auch inhaltlich zusammengehören. Am Beispiel der Tabellen „Kunden“ und „Bestellungen“ bedeutet dies, dass aus dem Kreuzprodukt nur die Zeilen angezeigt werden sollen, bei denen die Kundennummer der Tabelle „Bestellungen“ auch mit der Kundennummer der Tabelle „Kunden“ übereinstimmt. Dieser Sachverhalt

20 11.2. Numerische Datentypen (2010), http://dev.mysql.com/doc/refman/5.1/de/numeric-types.html

wird als Gleichung formuliert, weswegen diese Art des Joins als „Equi-Join“ bezeichnet wird. Der Join kann sowohl implizit als auch explizit formuliert werden, wie die folgenden zwei Beispiele zeigen:

- Explizit: SELECT * FROM Kunden INNER JOIN Bestellungen On `Kunden.Kunden-Nummer` = `Bestellungen.Kunden-Nummer`
- Implizit: SELECT * FROM Kunden, Bestellungen WHERE `Kunden.Kunden-Nummer` = `Bestellungen.Kunden-Nummer`

Intern löst der SQL-Server dieses Problem nicht über das aufwendige Kreuzprodukt, aus welchem dann die relevanten Zeilen gesucht werden, sondern verwendet dazu besser geeignete Methoden.

Ein SELECT kann auch eine „ORDER BY“-Klausel enthalten. Wird die „ORDER BY“-Klausel nicht angegeben, werden die selektierten Datensätze in der Reihenfolge wiedergegeben, in welcher der MySQL-Server sie intern gespeichert hat. Soll jedoch das Ergebnis nach einer oder mehreren Spalten sortiert wiedergegeben werden, so muss die „ORDER BY“-Klausel vorhanden sein. Diese übernimmt einen oder mehrere Spaltennamen, nach denen das Ergebnis sortiert wird.

Bei vielen SELECT-Abfragen soll nur ein Teil der mit WHERE spezifizerten Datensätze weiter verarbeitet werden. Um die Ergebnismenge einzuschränken wird das LIMIT-Element verwendet. Dieses übernimmt zwei Parameter: wieviele Datensätze angezeigt werden sollen und ab welchem Datensatz die Anzeige beginnen soll. Viele Programmierer verwenden dies um Listen innerhalb einer Webseite auf mehrere Seiten aufzuteilen. Auch kann die Klausel bei UPDATE und DELETE Statements eingesetzt werden, was in der Praxis verwendet wird um sicherzustellen, dass der Befehl nur die gewünschte Anzahl an Datensätzen ändert.

2.10.3 Schreibende Befehle: INSERT und UPDATE

Der INSERT-Befehl wird dazu verwendet um einen oder mehrere Datensätze in eine Tabelle einzufügen. Eine einfache Variante dieses Befehls lautet *„INSERT INTO `Tabellen-Name` VALUES ('1','2','3')“,* wobei 1, 2 und 3 die einzufügenden Daten sind. Ist z.B. die erste Spalte ein Primärschlüssel[21] und bereits ein Datensatz mit '1'

21 Primärschlüssel werden in Kapitel 2.10.4 „Primärschlüssel, Fremdschlüssel“ erläutert

vorhanden, so schlägt der INSERT-Befehl fehl, weil Primärschlüssel eindeutig sein müssen. Da neue Daten in die Tabelle eingefügt werden, handelt es sich bei INSERT um eine schreibende Operation.

Mit UPDATE lassen sich einzelne oder mehrere Datensätze einer Tabelle aktualisieren. Die grundlegende Syntax dafür lautet *„UPDATE `Tabellen-Name` set `spalte` = 'wert'".* Dieser Befehl würde alle Werte der Spalte nun auf denselben Wert setzen, sofern nicht die WHERE-Klausel wie bei SELECT zur Spezifizierung der zu ändernden Datensätze verwendet wird. UPDATE überschreibt vorher vorhandene Werte und ist wie INSERT eine datenverändernde und somit schreibende Datenbank-Operation.

2.10.4 Primärschlüssel, Fremdschlüssel

Oft ist es notwendig, dass jeder Datensatz innerhalb einer Tabelle eindeutig identifizierbar sein muss. Dies ist wichtig, falls der Datensatz geändert, gelöscht oder gezielt mit SELECT angezeigt werden soll. Sofern die Daten selbst keine eindeutige oder geeignete Identifizierung enthalten, kann hierfür ein künstlicher Primärschlüssel eingeführt werden. Am Beispiel des Shops wird so eine Kundennummer generiert, wenn sich ein neuer Kunde registriert. Zwar lässt sich ein Kunde über Vorname, Nachname und Geburtsjahr meist eindeutig identifizieren, jedoch ist es schwieriger den Kunden nur anhand dieser Kriterien in der Datenbank zu finden. Wenn dieser Kunde nun eine Bestellung auslöst, so muss zu der Bestellung in der Datenbank der entsprechende Kunde gespeichert werden. Ohne künstlichen Primärschlüssel müsste man den Kunden über Vorname, Nachname, Geburtsjahr und Adresse in der Bestellung identifizieren. Wenn der Kunde sein Geburtsjahr ändert, müssen diese Angaben dann aktualisiert werden und verbrauchen unnötig Speicherplatz. Besitzt der Kunde dagegen eine Kundennummer, so muss zur Verknüpfung der Bestellung mit dem Kunden nur diese Nummer in der Bestellung gespeichert werden. Man spricht von einem Fremdschlüssel, über welchen die Beziehung zwischen den beiden Tabellen aufgelöst werden kann.

Bei der Verwendung von Primärschlüsseln sorgt die Datenbank selbstständig dafür, dass dieser auch innerhalb einer Tabelle eindeutig ist und verhindert das Einfügen von mehr als einem Datensatz mit derselben Primärschlüsselnummer. Auch kann die

Datenbank bei Fremdschlüsseln überwachen, ob der referenzierte Datensatz wirklich existiert und verhindern das dieser gelöscht wird, solange noch Referenzen darauf bestehen. Die Datenbank erhält somit die referentielle Integrität.

Primärschlüsselspalten können in MySQL das Attribut „*Auto_Increment*“ (engl. für automatisches hochzählen) erhalten. Dies bewirkt, dass der Standardwert der Spalte vom MySQL-Server automatisch bestimmt wird. Bei einer neu erstellten oder geleerten Tabelle erhält so der erste eingefügte Datensatz automatisch den Primärschlüssel 1, der zweite 2 und so weiter. Oft wird zum Aufbau von Fremdschlüsselbeziehungen der durch ein vorangegangenes INSERT vergebene Primärschlüssel benötigt, so dass dieser in einer anderen Tabelle verwendet werden kann. Dieser Wert wird von der SQL-Funktion „LAST_INSERT_ID()“ und der C-API-Funktion „mysql_insert_id()“ zurückgegeben.

2.10.5 Tabellen-Sperren (Locks)

In der Regel ist es so, dass mehrere Applikationen oder mehrere Threads einer Applikation nahezu gleichzeitig eine Tabelle verwenden. Dabei entsteht das Problem, dass die Konsistenz der Datenbank verloren gehen kann oder nur teilweise geschriebene und damit ungültige Daten gelesen werden. Um dies zu verhindern, setzt MySQL sogenannte „Locks“ (engl. für Schlösser, Sperren) ein. Diese Sperren erlauben einzelnen schreibenden Operationen exklusiven Zugriff auf eine Tabelle, so dass andere lesende oder schreibene Operationen blockiert werden, bis die Arbeit abgeschlossen und die Sperrung wieder aufgehoben ist. Sofern es sich nicht um eine transaktionsfähige Datenbank-Engine wie InnoDB oder Berkely DB handelt, verwendet MySQL automatisch tabellenbasierende Sperren.[22] Dass bei konkurrierenden Datenbankabfragen die gesamte Tabelle und nicht nur die betroffenen Teile gesperrt werden, liegt in der Performance begründet: Es ist meist besser, die gesamte Tabelle zu sperren als einzelne Zeilen, da eine tabellenweite Sperrung schneller auf- und wieder abgebaut werden kann, was vorallem bei Tabellen mit sehr vielen Datensätzen von Bedeutung ist.

22 Oracle Corporation, MySQL-Referenzhandbuch, 7.7.2. Table Locking Issues (2010) http://dev.mysql.com/doc/refman/5.0/en/table-locking.html

3 Problemanalyse

3.1 Bisheriger Zustand

Wie schon erwähnt, besitzt das Unternehmen z.Z. zwei Server. Im folgenden werden diese als Server 1 und Server 2 bezeichnet. Bei beiden Systemen handelt es sich um handelsübliche Hardware, auf welchen ein 64-Bit ArchLinux als Betriebssystem installiert ist. Bei den Prozessoren handelt es sich um AMD-Opteron-Prozessoren mit 32/64-Bit Unterstützung. Beide Server befinden sich im selben Rechenzentrum, aber nicht im selben Subnetz. Die Latenz zwischen beiden Servern bewegt sich LAN-typisch zwischen 0,2 und 0,4 Millisekunden.

Auf beiden vom Unternehmen eingesetzten Servern laufen neben jeweils einem Apache-Webserver und einer MySQL-Datenbank auch ein Postfix-E-Mailserver, wobei sich der E-Mailspeicher auf Server 2 befindet. Server 1 agiert als Backup-E-Mailserver und leitet alle eingehenden E-Mails an Server 2 weiter. Neben den genannten Diensten stellen die Server noch weitere, weniger kritische, Anwendungen wie FTP, SSH und SVN zur Verfügung, welche im Weiteren nicht mehr erwähnt werden, da sie für die Aufgabenstellung irrelevant sind.

Die Kundenprojekte sind auf beide Server aufgeteilt und verwenden den Webserver und die Datenbank sowie den lokalen E-Mail-Server zum Versenden von E-Mails. Die Nutzdatenmenge beträgt auf beiden Systemen zusammen zur Zeit ca. 50GB und beide Server sind in Stoßzeiten stark ausgelastet, wobei die Auslastung auf Server 1 überwiegt.

Zu den Kundenprojekten gehören auch Cron-Jobs, die PHP-Skripte und C++-Programme ausführen, welche mit Schnittstellen von Drittanbietern kommunizieren oder Arbeiten am lokalen Datenbestand durchführen.

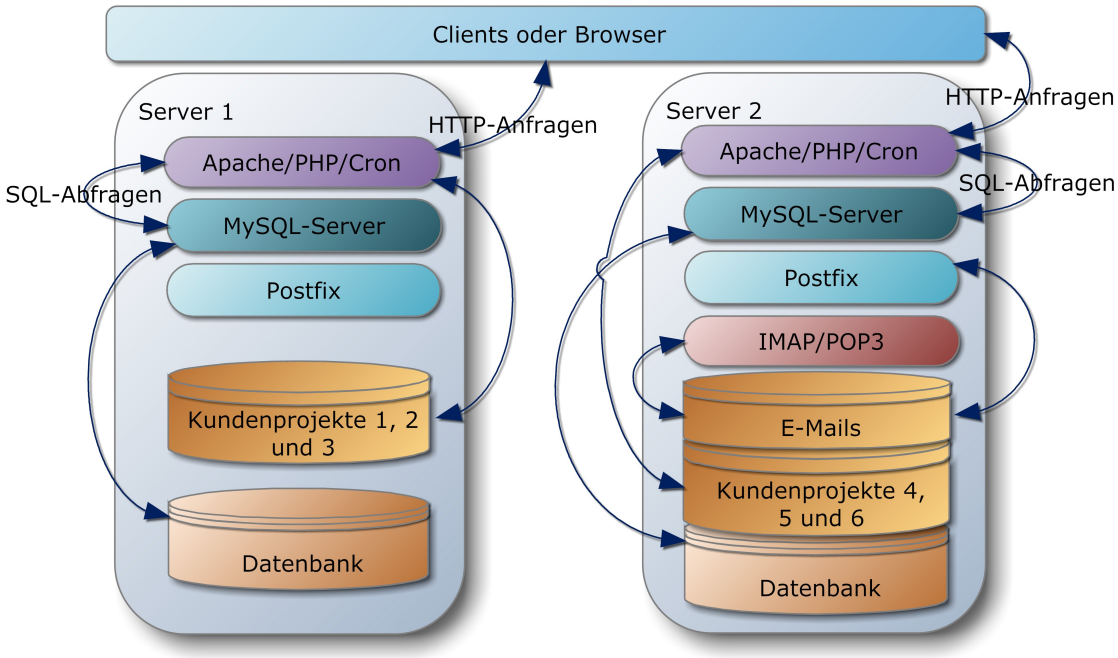

Abbildung 3: Vereinfachte Darstellung der Ausgangssituation - Apache, PHP und die Cron-Jobs wurden zusammengefasst, irrelevante Dienste weggelassen

Quelle: eigene Abbildung

3.2 Wichtige Daten

Da das Ziel die Erarbeitung einer Hochverfügbarkeitslösung für die angebotenen Dienste ist, ist es wichtig zu wissen, welche Daten in Zukunft ggf. nicht nur redundant, sondern auch synchron gehalten werden müssen. Außerdem ist die Beschaffenheit der Daten zu beachten. Tabelle 2 stellt eine Übersicht der redundant zu haltenden Daten dar.

Name	***Beschreibung***	***Ort, Pfad***	***Relevanz***
Kunden-dokumente	Alle Dateien der Kundenprojekte; größtenteils statische Dateien wie Bilder, PHP-Skripte und C++ Programme aber auch sich stetig ändernde Dateien wie Logfiles und Zwischenergebnisse der ausgeführten Cron-Jobs	Server 1 Server 2 /home/www	Essenziell, da ohne diese die Webpräsenzen nicht verfügbar sind
MySQL-Datenbank	Die zu den Kundenprojekten gehörigen Datenbanken. Diese sind teilweise sehr hoch frequentiert. Im Falle eines Serverausfalls muss die Datenbankintegrität gewahrt bleiben.	Server 1 Server 2 /var/lib/mysql	Essenziell
E-Mails	Kunden-E-Mails. Die meisten Kunden rufen die E-Mails via POP3 ab, einige lassen sie aber auf dem Server gespeichert.	Server 2 Dateisystem /home/vmail	Mittel Bei einem Server-ausfall wäre eine Nicht-verfügbarkeit der E-Mail-Postfächer für einige Minuten vertretbar

Tabelle 2: Redundant zu haltende Daten

Quelle: eigene Zusammenstellung

3.3 Anforderungen

In diesem Kapitel werden die Anforderungen an die Hochverfügbarkeitslösung betrachtet, auf deren Basis anschließend die Entscheidung für eine konkrete Implementierung getroffen wird. Diese Lösung soll möglichst viele der Anforderungen erfüllen, welche vom Unternehmen vorgegeben wurden.

3.3.1 Verfügbarkeit und Flexibilität bei der Reaktionszeit

Selbstverständlich ist eine hohe Verfügbarkeit des Gesamtsystems die Hauptanforderung. Daher sollte es keine *Single Points of Failure* (Deutsch: einzelne Stellen des Scheiterns) geben. Der entstehende Schaden bei einem Serverausfall soll so gering wie möglich gehalten werden, wobei natürlich auch die Nichterreichbarkeit der Dienste als Schaden gewertet wird.

Die Anforderungen an die Verfügbarkeit sind im vorliegendem Fall vermutlich nicht so hoch, wie sie bei Web-Diensten mit tausenden Besuchern pro Minute sind. Bei den behandelten Webshops ist eine Nichtverfügbarkeit von einigen Sekunden bis Minuten zwar ärgerlich, aber kein Grund für merklich hohe Umsatzeinbußen des jeweiligen Händlers. Wie es oft der Fall ist, muss auch hier ein Gleichgewicht zwischen dem potentiell durch einen Ausfall entstehenden Schaden und den Kosten zur Vermeidung des Ausfalls bestehen.

Die Lösung soll bei einem Serverausfall kein sofortiges Handeln erfordern. So kann z.B. bei einem Serverausfall außerhalb der Geschäftszeiten die Bearbeitung zu den regulären Arbeitszeiten erfolgen. Alle Dienste sollen weiter verfügbar sein, ohne dass dafür manuell irgendwelche Aktionen getätigt werden müssen. Nach Möglichkeit soll sich das System bei Wiederverfügbarkeit des ausgefallenen Servers automatisch regenerieren können.

3.3.2 Freie Open-Source-Software

Die bei der Lösung einzusetzende Software soll freie Open-Source-Software sein. Dies ist der Wunsch des Unternehmens, da dieses bisher ausschließlich freie Open-Source-Software auf ihren Servern einsetzt, und dies auch in Zukunft beibehalten möchte. Dies ist nicht nur aus Gründen der Kostenersparnis gegenüber kommerziellen Lösungen, sondern auch aufgrund von Flexibilität und Kompatibilität gewünscht. Leider kommt es vor, dass selbst properietäre Software, hinter der große Unternehmen stehen, mit der Entwicklungsgeschwindigkeit von Linux nicht mithalten kann, so dass wichtige Updates ggf. sehr spät erscheinen. Auch besteht immer die Gefahr, dass der Hersteller die Unterstützung für das Produkt einstellt. Freie Software dagegen kann von den Nutzern weiter entwickelt werden. Außerdem bietet Open-Source-Software immer die Möglichkeit, diese selbstständig zu erweitern und benötigte Features mit mehr oder weniger Aufwand nachzurüsten, was einen Wettbewerbsvorteil bedeuten kann.

3.3.3 Keine speziellen Hardwareanforderungen

Es soll, wenn möglich, die bereits vorhandene Hardware genutzt werden, um die Problemstellung zu lösen. Dabei handelt es sich um angemietete Server, welche als Komplettpaket mit Anbindung, Traffic, Strom und Hardware für ca. 70-150€ pro Stück und Monat gemietet werden können und sowohl Unternehmen als auch Privatkunden schnell und unkompliziert zur Verfügung stehen.

Dies bietet viele Vorteile gegenüber dem Hosting in der eigenen Firma oder der Verwendung von eigener Hardware im Rechenzentrum: Der Server-Anbieter übernimmt alle organisatorischen Pflichten, stellt den Server meist kostenlos auf, tauscht schnell und ohne Diskussion defekte Teile oder stellt bei Totalausfall ohne Mehrkosten einen komplett neuen Server zur Verfügung. Außerdem besteht am Ende der Vertragslaufzeit immer die Möglichkeit, einen neuen Vertrag mit neuer Server-Hardware abzuschließen. Wechselt man so die Server-Hardware am Ende der Vertragslaufzeit, wird auch die Wahrscheinlichkeit, dass eine Festplatte oder ein an Alterserscheinungen leidendes Netzteil ausfällt geringer.

Soll dagegen eigene Hardware aufgestellt werden, so muss diese eigenständig gekauft, bezahlt, verschickt und versichert werden. Außerdem ist man bei Ausfall selbst dafür verantwortlich, schnell Ersatz zu beschaffen. Beim Tauschen der Hardware können außerdem zusätzliche Kosten für Anfahrt oder einen Techniker des Rechenzentrums entstehen. Auch ist die Miete für den Platz im Rechenzentrum und die dazugehörige Anbindung für eigene Hardware meist allein schon teurer, als die entsprechenden Komplettangebote des Serveranbieters.

Da bei den genannten günstigen Komplettlösungen nur die Auswahl zwischen verschiedenen Server-Paketen besteht und spezielle Hardwarekomponenten selten nachgerüstet werden können, soll die Lösung mit dieser weit verbreiteten Standard-Hardware auskommen.

3.3.4 Effektive Ressourcennutzung und beibehalten der Webanwendungen

Ressourcen wie CPU-Leistung, Festplattenkapazität und Netzwerktraffic waren schon immer knappe Güter. Auch bei ständig steigender Leistungsfähigkeit der Hardware hat sich daran nichts geändert, denn ein Webserver kann selten schnell genug sein. Für kleine Projekte mit etwa 1000 Besuchern pro Tag stellt ein einzelner Webserver meist noch ausreichend Ressourcen zur Verfügung, um nicht nur die Nutzerabfragen, sondern auch die im Hintergrund laufenden Backup-Dienste, Importe sowie Exporte in angemessener Zeit zu bearbeiten. Wird das Projekt immer populärer, steigen etwa proportional zu den Besucherzahlen auch die Anforderungen an die Hardware-Ressourcen. Eine der wichtigsten Anforderungen an die Hochverfügbarkeitslösung ist somit, die zur Verfügung stehende Hardware möglichst effizient zu nutzen, um wirtschaftlich zu sein. Einige Lösungen benötigen die doppelte Hardware und bieten dabei keinen Leistungsvorteil. Diese sind gegenüber einer Lösung, welche sowohl Leistung und Verfügbarkeit erhöht im Nachteil.

Die neue redundante Hosting-Plattform soll außerdem die bereits vorhandenen Webapplikationen möglichst ohne große Veränderungen aufnehmen. Zwar ist für alle in der Firma eingesetzen Webanwendungen der Source-Code verfügbar, aber zu umfangreiche Anpassungen wären nicht nur zusätzlicher Programmieraufwand, sondern könnten die Pflege der Software verkomplizieren.

3.3.5 Horizontale Skalierbarkeit

Bei neuen Lösungen für IT-Probleme ist es nicht nur wichtig, dass diese die aktuellen Leistungsanforderungen ausreichend abdecken, sondern auch bei einer Verdopplung oder Verzehnfachung der Anforderungen noch skalierbar sind. Skalieren kann man sowohl indem man einen immer leistungsfähigeren Server einsetzt, oder aber, in dem man nicht bessere sondern einfach mehr Server hinzufügt. Das Austauschen eines zu langsamen Servers gegen einen schnelleren bezeichnet man als „vertikale Skalierung“, das Hinzufügen von mehreren Servern dagegen als „horizontale Skalierung“. Die meisten Anwendungen sind problemlos vertikal, also durch den Einsatz leistungsfähigerer Hardware, skalierbar, wobei dem schnell Grenzen gesetzt

sind. Siehe dazu Abbildung 4. Es existieren zwar Server-Systeme, die deutlich leistungsfähiger sind als die herkömmliche Standard-Hardware[23], dadurch aber nicht-linear teurer. Dagegen ist das Hinzufügen von mehr Systemen günstiger und weiter skalierbar, sofern die verwendete Software dies erlaubt. Mit „Skalierbarkeit“ ist somit meist die horizontale Skalierbarkeit gemeint.

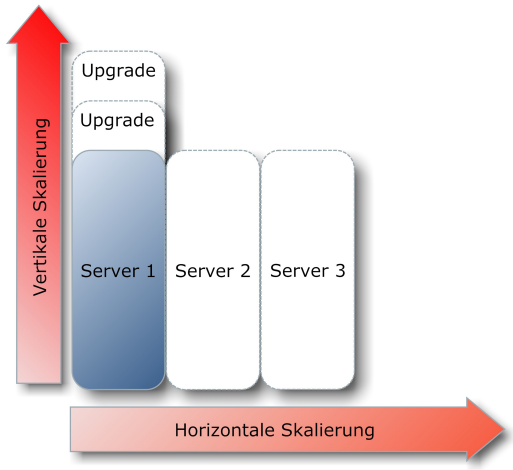

Abbildung 4: Horizontale und vertikale Skalierbarkeit

Quelle: eigene Abbildung

4 Theoretische Konzepte für Hochverfügbarkeits-LAMP

Es existieren verschiedene Konzepte, mit denen sich eine LAMP-Umgebung hochverfügbar gestalten lässt. In diesem Kapitel werden ganz grundlegende Ansätze zur Realisierung der Aufgabenstellung betrachtet, wobei nicht auf eine spezifische Implementierung der Lösung eingegangen wird. Die meisten Konzepte benötigen einen gemeinsamen Speicher und ggf. eine Clusterverwaltungssoftware, die den Ausfall eines Knotens erkennt und entsprechende Maßnahmen einleitet. Ein gemeinsamer Speicher ist hierbei ein Speichermedium, welches allen beteiligten Servern gleichermaßen zur Verfügung steht. Es müssen sowohl bei einem Aktiv/Aktiv-Cluster als auch bei einem Aktiv/Passiv-Cluster die Zugriffe verteilt bzw. umgeleitet werden. Wie diese Vorraussetzungen realisiert werden, ist nicht Bestandteil der allgemeinen theoretischen Betrachtung in diesem Kapitel, weswegen dies nicht immer explizit erwähnt wird.

4.1 Einfaches applikationsbasierendes Failover mit gemeinsamen Speicher

Alle relevanten Daten werden auf einem gemeinsamen Speichergerät gespeichert. Der aktive Server arbeitet mit diesen Daten. Bei einem Ausfall des Servers aktiviert der

23 Siehe Abschnitt 3.3.3 - Keine speziellen Hardwareanforderungen

passive Server seinerseits das gemeinsame Speichergerät und startet alle Dienste, die auf die Daten zugreifen. Alle Zugriffe werden nun auf den passiven Server umgeleitet und die Website ist wieder erreichbar.

In unserem Beispiel bedeutet dies, dass HTTP-Dokumente, MySQL-Datenbankdateien und E-Mails auf dem gemeinsamen Speicher liegen, auf welche Server 1 exklusiv zugreift. Fällt Server 1 aus, so startet Server 2 Web- E-Mail- und Datenbankserver unter Verwendung der bereitgestellten Daten. Anschließend müssen alle Zugriffe auf Server 2 umgeleitet werden.

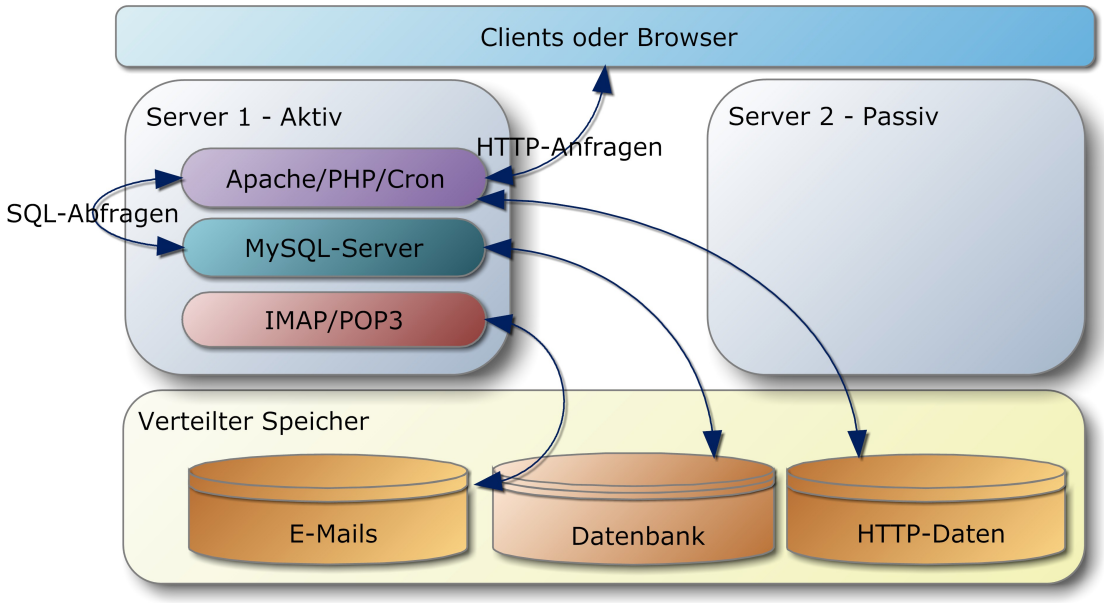

Abbildung 5: Ausgangszustand: Einfaches applikationsbasierendes Failover mit gemeinsamen Speicher

Quelle: eigene Abbildung

4.2 Failover mit virtuellen Maschinen

Da ein Betriebssystem mit allen Anwendungen innerhalb einer virtuellen Maschine läuft, kann dieses auch von System zu System übertragen werden, vorrausgesetzt, dass das Zielsystem eine kompatible Virtualisierungsplattform zur Verfügung stellt. Es liegt nahe, diesen Vorteil zu nutzen, um bei ausfallender Hardware die Software auf einem anderen System weiter zu verwenden. Dabei werden alle wichtigen Dienste und

Nutzdaten innerhalb der virtuellen Maschine eingerichtet.

4.2.1 Einfaches Failover mit virtuellen Maschinen

Dies ist das einfachste Setup mit virtuellen Maschinen. Beide Server besitzen die Möglichkeit, virtuelle Maschinen zu starten und teilen sich einen gemeinsamen Speicher, auf welchem die virtuelle Maschine gespeichert ist. Im Ausgangspunkt wird eine virtuelle Maschine mit allen benötigten Diensten auf Server 1 ausgeführt. Fällt Server 1 aus, so wird diese virtuelle Maschine unter Verwendung der bereitgestellten virtuellen Festplatte auf Server 2 gestartet. Diese virtuelle Maschine bietet ab da alle Dienste, die zuvor auf Server 1 verfügbar waren, nun auf Server 2 an.

Der Nachteil des einfachen Failovers mit virtuellen Maschinen liegt auf der Hand: Wenn Server 1 offline geht, gehen alle im Arbeitsspeicher befindlichen Informationen verloren. Dabei kann es passieren, dass Dateien nur teilweise geschrieben wurden und somit nun defekt sind, analog zu einem normalen Computerabsturz. Die meisten Linux-Serverdienste sind auf diesen Fall vorbereitet und sorgen dafür, das ein Absturz während des Betriebes keine weitreichenden Folgen hat. Dennoch lässt sich dies nicht für alle Anwendungen garantieren.

Der Vorteil liegt darin, dass im Falle eines Hardwaredefekts die virtuelle Maschine mit allen Diensten schnell wieder gestartet werden kann, ohne das vorher die Hardware repariert oder ein Backup wiederhergestellt werden muss. Ein gleichzeitiges Starten der virtuellen Maschine auf beiden Servern wäre allerdings fatal und muss verhindert werden, da sonst die Datenintegrität gefährdet ist.[24]

4.2.2 Hochverfügbarkeit mit Live-Migration

Es gibt viele Gründe eine virtuelle Maschine von einem Host-System zu einem anderen zu migrieren, beispielsweise die Installation eines neuen Kernels, der Wechsel einer defekten Festplatte oder gar der Austausch des gesamten Host-Systems. Klassischerweise wird die virtuelle Maschine heruntergefahren, ggf. auf das neue Host-System kopiert und dort wieder gestartet. Dadurch entsteht natürlich eine Offlinezeit von ein paar Minuten.

24 Siehe dazu auch Kapitel 7.5.1.4 „Die Rolle des Arbitrators“. Die da beschriebene Split-Brain-Problematik trifft auch auf viele andere Hochverfügbarkeitslösungen zu

Die Virtualisierung der Hardware bietet hierbei aber eine praktische Möglichkeit: die sogenannte „Live-Migration“. Dabei wird die bereits gestartete virtuelle Maschine in ihrem aktuellen Zustand von einem Host auf einen anderen übertragen. Dafür ist meist Vorraussetzung, dass sich die virtuelle Maschine auf einem gemeinsamen Speicher befindet. Dann kann der Hypervisor angewiesen werden, mit der Übertragung der virtuellen Maschine zu beginnen, wobei er den gesamtem Arbeitsspeicher und den Zustand der virtuellen Hardware incl. der CPU auf den anderen Host überträgt. Anschließend wird die virtuelle Maschine auf dem neuen Host fortgesetzt.

Die Grundvorraussetzung für diese Live-Migration ist, dass die IP-Adresse der virtuellen Maschine auch nach dem Umzug auf einen anderen Server unverändert und für alle erreichbar bleibt. Befindet sich der Zielserver in einem anderen Netzwerksegment, in dem keine Route zu der IP der virtuellen Maschine führt, so ist diese dann über diese IP nichtmehr erreichbar.

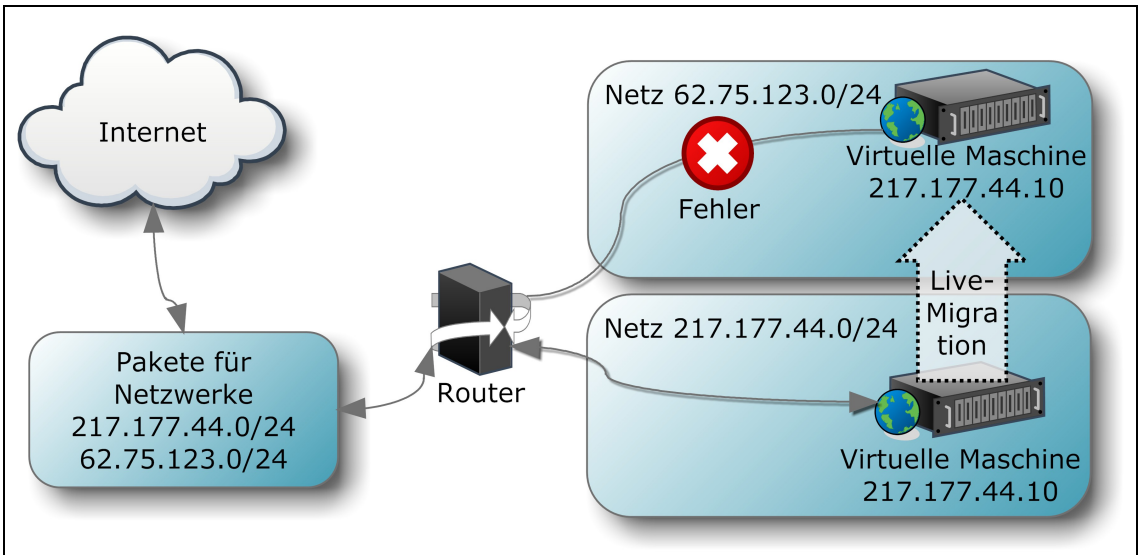

Abbildung 6: Migration einer virtuellen Maschine in ein anderes Netz

Quelle: eigene Abbildung

Dies ist als Lösung bei geplanten Wartungsarbeiten anwendbar, bietet aber leider bei ungeplanten Serverausfällen keine Vorteile. Um eine virtuelle Maschine zu migrieren, muss diese und deren Host-System noch aktiv sein, was bei einem plötzlichen unerwarteten Hardwaredefekt nicht der Fall ist. Zwar lässt sich ein Hardwaredefekt in manchen Fällen z.B. mittels SMART-Daten der Festplatte vorraussagen, jedoch ist

dies allein nicht ausreichend.

4.2.3 Hochverfügbarkeit mit synchronisierten virtuellen Maschinen

Das Problem der Live-Migration ist, dass die virtuelle Maschine auf den neuen Server übertragen werden muss, ehe der ursprüngliche Server ausfällt. Da der tatsächliche Ausfallzeitpunkt nicht genau vorrausgesagt werden kann und ein Ausfall sich auch nicht ankündigen muss, ergibt sich daraus der Ansatz der synchronisierten virtuellen Maschinen. Prinzipiell bedeutet dies, dass die virtuelle Maschine so früh wie möglich auf den zweiten Server migriert wird. Doch anstatt die Migration abzuschließen und die virtuelle Maschine auf dem Ursprungsserver zu beenden läuft diese weiterhin. Ab diesem Zeitpunkt werden alle neu entstehenden Änderungen der virtuellen Maschine vom Ursprungssystem auf den zweiten Server übertragen. Fällt der erste Server aus, wird die Migration abgeschlossen und der zweite Server übernimmt nahtlos. Dabei müssen sich wieder beide Server im selben Subnetz befinden.

4.3 Hochverfügbarkeit auf Anwendungsebene mit gemeinsamem Speicher und Lastverteilung

Die meisten Webanwendungen sind so beschaffen, dass sie von mehreren Nutzern gleichzeitig verwendet werden können. So werden beispielsweise Dateien wie Nutzerbilder erst auf dem Server in ein temporäres Verzeichnis hochgeladen. Anschließend werden sie an die gewünschte Stelle verschoben und daraufhin nach erfolgreichem Schreiben mit Hilfe der Datenbank referenziert, so dass die Datei für andere Nutzer sichtbar wird. Der Webserver bearbeitet derartige Vorgänge für jeden Nutzer in einem unabhängigen Verarbeitungsstrang, weshalb diese Vorgänge zwangsläufig so ausgelegt sein müssen, dass sie parallel ausgeführt werden können. Diese Verarbeitungsstränge arbeiten letztendlich mit demselben Dateisystem, weswegen es problemlos machbar ist, die meisten Webanwendungen mit mehreren Webservern gleichzeitig mit demselben Dateisystem zu betreiben. Dies ermöglicht die Verteilung aller Anfragen auf zwei Server, welche beide in der Lage sind, diese zu bearbeiten. Wenn die Anfragen bei Ausfall eines Servers an den anderen umgeleitet werden, ergibt sich daraus eine bessere Verfügbarkeit der Webanwendungen.

Obwohl dies bei mehreren Webservern mit einem gemeinsamen Speicher problemlos möglich ist, wird es mit mehreren Datenbankservern schwierig: Die Datenbankserverdateien werden vom jeweiligen Datenbankserver in der Regel exklusiv gelesen, geschrieben und verwaltet. Ein gleichzeitiges Starten von 2 Datenbankservern mit den selben Datenbankdateien ist somit unmöglich – zumindest wenn die Daten nach einigen schreibenden Abfragen noch konsistent sein sollen.

Da Datenbankserver in der Regel über das Netzwerk erreichbar sind und somit die Webanwendung nicht zwangsläufig auf demselben Server wie die Datenbank ausgeführt werden muss, wäre es eine Möglichkeit den Datenbankserver nur auf einem System laufen zu lassen. Bei einem Ausfall von Server 1 wie im Kapitel 4.1 „Einfaches applikationsbasierendes Failover mit gemeinsamen Speicher“ beschrieben, wird der Datenbankserver auf Server 2 unter Verwendung der Datenbankdateien auf dem gemeinsamen Speicher wieder gestartet. Da der Leistungsengpass bei den meisten Webanwendungen weniger der Webserver als viel mehr die Datenbank ist, bietet dies so gesehen keine Vorteile gegenüber dem einfachen Failover.

Es bestehen kaum Chancen, die Datenbank einer Webanwendung zu skalieren, wenn der Datenbankserver dafür keine Möglichkeiten bietet.

4.4 Tabellarische Zusammenfassung der Konzepte

Konzept	***Vorteile***	***Nachteile***
Einfaches applikationsbasierendes Failover mit gemeinsamen Speicher	• Einfacher Aufbau, wenig Aufwand	• Bei ungeplanten Ausfällen ggf. Datenverlust • Datenbankdateien müssen ggf. repariert werden • Nicht skalierbar
Allgemein für alle auf virtuellen Maschinen basierenden Lösungen	• Mehr Sicherheit durch Isolierung der einzelnen Dienste • Virtuelle Maschinen ohne Änderungen auf einen anderen Host transferierbar	• Weniger Performance durch Emulations-Overhead • Höherer administrativer Aufwand zur Wartung des Hosts und aller Gäste • Nicht skalierbar

Konzept	***Vorteile***	***Nachteile***
Einfaches Failover mit virtuellen Maschinen	• große Auswahl bei der Virtualisierungs-Software	• Bei ungeplanten Ausfällen ggf. Datenverlust • Bei geplanten Wartungen und ungeplanten Ausfällen Offlinezeit, da virtuelle Maschine neustarten muss • Datenbankdateien müssen ggf. repariert werden
Hochverfügbarkeit mit Live-Migration	• Effektiv bei geplanten Wartungen • Zusätzlich zu den geforderten LAMP-Diensten lassen sich damit auch viele andere Dienste ausfallsicher gestalten	• Bei ungeplanten Ausfällen kein Vorteil gegenüber einfachem Failover mit virtuellen Maschinen • Alle Host-Server müssen sich im selben Netz befinden
Hochverfügbarkeit mit synchronisierten virtuellen Maschinen	• Sowohl bei geplanten als auch bei ungeplanten Ausfällen effektiv • Kein Datenverlust, nahtlose Fortsetzung auf einem anderen System • Zusätzlich zu den geforderten LAMP-Diensten lassen sich damit auch viele andere Dienste ausfallsicher gestalten	• Alle Host-Server müssen sich im selben Netz befinden, damit Netzwerk-verbindungen nicht getrennt werden
Hochverfügbarkeit auf Anwendungsebene mit gemeinsamem Speicher und Lastverteilung	• Effiziente Nutzung der Hardwareressourcen durch Verteilung der Last • Horizontale Skalierbarkeit	• Nur für Webanwendung und ggf. andere datenbankbasierende Anwendungen vorteilhaft • Bei Ausfällen ggf. Daten-verlust

Tabelle 3: Zusammenfassung Vor- und Nachteile

4.5 Fazit

Es kamen zwei grundsätzlich verschiedene Lösungen in die engere Auswahl: Zum einen die applikationsunabhängige Lösung 4.2.3 „Hochverfügbarkeit mit

synchronisierten virtuellen Maschinen“, zum Anderen die applikationsabhängige Lösung 4.3 „Hochverfügbarkeit auf Anwendungsebene mit gemeinsamem Speicher und Lastverteilung“. Das Ergebnis ergibt sich letztendlich aus der Wichtung der zuvor in Kapitel 3.3 „Anforderungen“ ermittelten Kriterien. Wäre nur die hohe Verfügbarkeit der bisherigen Anwendungen wichtig, würde die virtuelle Maschine die günstigste Variante darstellen. Diese ermöglicht neben der Isolierung der Dienste durch die Live-Migration und Synchronisierung eine sehr hohe Verfügbarkeit mit fast nicht wahrnehmbarer Ausfallzeit. Leider ist es aber so, dass nicht nur die Verfügbarkeit, sondern mit wachsenden Besucherzahlen und Leistungsanforderungen auch die Leistungsfähigkeit ansteigen muss.

Eine virtuelle Maschine kann maximal so leistungsfähig sein, wie der physikalische Rechner, auf welchem sie ausgeführt wird. Reicht die Leistungsfähigkeit des Rechners nicht mehr aus, um der virtuellen Maschine und den ausgeführten Diensten genügend Ressourcen zur Verfügung zu stellen, muss erneut eine Lösung gefunden werden. Diese muss nicht nur hochverfügbar, sondern auch skalierbar sein. Ist dies nicht gegeben, bedeutet das in der Praxis bei hohem Besucheraufkommen ein Erliegen des Webshops, so dass ggf. keine einzige Bestellung mehr aufgegeben werden kann. In diesem Fall wäre der Erfolg des Webshops gleichzeitig dessen Ende.

Wünscht man neben hoher Verfügbarkeit auch Skalierbarkeit, so bleibt nur noch die Möglichkeit, dies applikationsbasierend, wie in Kapitel 4.3 beschrieben zu realisieren. Welche Möglichkeiten sich konkret anbieten hängt zum Einen von der verwendeten Datenbankverwaltungssoftware, zum Anderen von der Beschaffenheit der Webapplikation ab.

Auch wenn das Konzept der synchronisierten virtuellen Maschinen nicht weiter betrachtet wird, kann später durchaus eine Kombination mit diesem möglich sein. So können beispielsweise einzelne Serverdienste, die selbst keine hohen Leistungsanforderungen besitzten, oder auch alle Cron-Jobs, die bei Serverausfall nicht unterbrochen werden sollen, innerhalb einer virtuellen Maschine ausgeführt werden. Von da aus können sie auf den hochverfügbaren Datenbankserver und den gemeinsamen Speicher zugreifen.

5 Vorrausetzungen: gemeinsamer Speicher, Umleitung der Anfragen

5.1 Gemeinsamer Speicher

Oft ist es notwendig, das Applikationen, die auf verschiedenen Servern laufen, Zugriff auf gemeinsam genutzte Daten erhalten. Abgeleitet vom englischen Begriff „*Shared Storage*“ wurde die Bezeichnung „gemeinsamer Speicher“ als Sammelbegriff für alle Technologien, die dieses Problem lösen, gewählt. In der Regel wird dies als Dateisystem implementiert, welches transparenten Zugriff auf die Dateien bietet.

Eine einfache Variante des gemeinsam nutzbaren Speichers ist eine Netzwerkfreigabe: Ein Server speichert alle Daten lokal und andere Server können über das Netzwerk darauf zugreifen. Drei unter Linux verfügbare Netzwerkdateisysteme dieser Art sind neben vielen anderen NFS (Network File System), CIFS (Common Internet File System, auch bekannt als SMB bzw. Windows Freigabe) und das auf SSH bzw. SFTP basierende Dateisystem SSHFS. Das Nutzen von nur auf einem Server gespeicherten Dateien bietet keinerlei Redundanz und bei Ausfall oder Nichterreichbarkeit des Servers sind die Daten für alle anderen nicht mehr verfügbar. Um Redundanz zu schaffen kann DRBD (**D**istributed **R**eplicated **B**lock **D**evice) verwendet werden – eine Software die mittlerweile Bestandteil des Linux-Kernels ist und ähnlich wie ein RAID 1 eine Spiegelung der Festplatte im Netzwerk realisiert. Das DRBD-Laufwerk kann mit jeden beliebigen Dateisystem formatiert und anschließend freigegeben werden. Fällt der Server, der die Freigabe hält, aus, so kann ein anderer den Exklusivzugriff auf das DRBD-Laufwerk erhalten, das Dateisystem einbinden und anschließend wieder freigeben. Dies ist nur ein möglicher Anwendungszweck für DRBD; in Kombination mit einem „Shared Disk File System“ wie Global File System (GFS) von Red Hat oder OCFS2 von Oracle, beide unter GPL, kann die DRBD-Festplatte gleichzeitig von mehreren Rechnern eingebunden werden.[25]

Ein gemeinsamer Speicher ist für die Realisierung der genannten Lösungen notwendig. Welche Art von gemeinsam nutzbaren Speicher letztendlich gewählt wird, ist weniger relevant, solange die Leistungsmerkmale, die Systemanforderungen und

25 Reisner, Philipp und Ellenberg, Lars, DRBD v8 - Replicated Storage with Shared Disk Semantics (2007), http://www.drbd.org/fileadmin/drbd/publications/drbd8_wpnr.pdf

die Ansprüche an Redundanz und Verfügbarkeit erfüllt werden. Ein genauer Vergleich und Test aller verfügbaren Netzwerkdateisysteme und sonstigen infrage kommenden Lösungen würde den Umfang dieser Thesis überschreiten. Für die spätere Realisierung im Unternehmen wird das fehlertolerante verteilte Dateisystem Ceph favorisiert. Zwar befindet sich dieses noch in Entwicklung und wird noch nicht für den produktiven Einsatz empfohlen, jedoch liegt die Vermutung nahe, dass sich dies durch die Aufnahme des Dateisystems in den Linux-Kernel der Version 2.6.34 bald ändern könnte. Welches Dateisystem letztendlich verwendet wird, entscheidet das Unternehmen, wenn die in dieser Thesis erarbeitete Lösung Mitte 2011 praktisch umgesetzt wird.

5.2 TCP/HTTP Lastverteilung / Anfragenumleitung

Eine der wichtigsten Fragestellungen bei der Planung einer jeden Hochverfügbarkeits- oder Lastverteilungslösung ist, wie die Clients auf die jeweiligen Server zugreifen. Fällt beispielsweise einer von zwei Servern einer Lastverteilungslösung oder der primäre Server eines Failover-Setups aus, so müssen alle anfragenden Clients ausschließlich auf den verbleibenden Server umgeleitet werden. Diese Umleitung muss möglichst schnell erfolgen, da sonst für Nutzer der Eindruck entsteht, das der Service nicht mehr verfügbar ist.

Dieses Kapitel befasst sich mit Lösungsansätzen, welche bei einer LAMP-Umgebung mit den geforderten Merkmalen zum Einsatz kommen können. Da der Kontakt mit dem Besucher einer Website zwischen Browser und Webserver entsteht, müssen die Anfragen auf dieser Ebene umgeleitet werden. Wird der Browser des Besuchers durch Anklicken eines Links oder die Eingabe des Domainnamens der Website dazu aufgefordert, eine Seite zu laden, wird zuerst die angeforderte Domain mittels DNS zu einer IP-Adresse aufgelöst. Die Auflösung eines Domainnamens zu einer IP geschieht rekursiv über mehrere DNS-Server. Der Browser fordert die Auflösung der Domain beim lokalen Betriebssystem an. Dieses sendet die Anfrage an den für das aktuelle Netzwerk definierten DNS-Server, von wo aus weitere, übergeordnete DNS-Server in die Bearbeitung der Anfrage involviert werden. Ist die Antwort gefunden, wird diese über das Betriebssystem an den Browser zurückgegeben. Ist die Ziel-IP bekannt, kann der Browser eine TCP-Verbindung anfordern, über welche er die HTTP-Anfrage

übermittelt und anschließend die entsprechende Antwort erhält.

Der einfachste Weg, die Browseranfragen umzuleiten, scheint das Ändern der Domain-Ziel-IP-Adresse zu sein, so dass der Browser statt der IP von Server 1 die von Server 2 erhält und dadurch die Verbindung zu diesem aufbaut. Warum dies problematisch ist und welche Alternativen es gibt wird in den folgenden Unterkapiteln erläutert.

5.2.1 Failover via DNS

Wie bereits erwähnt, erfolgt die Auflösung einer Domain über mehrere DNS-Server, wobei der Letzte in der Kette auch ein eigener, frei konfigurierbarer DNS-Server sein kann. Auf diesem kann die Ziel-IP der Domain beliebig geändert werden, beispielsweise durch eine Cluster-Verwaltungs-Software, welche den Ausfall des bisherigen Ziel-Servers erkennt. Die DNS-Server selbst sind in der Regel redundant vorhanden, so dass ein Ausfall der Namensauflösung unwahrscheinlich ist. Das Hauptproblem bei der Änderung der Ziel-IP ist, dass diese Änderung auch den anfragenden Browser erreichen muss. Da DNS-Abfragen vergleichsweise lange dauern, werden die einmal ermittelten Ergebnisse an verschiedenen Stellen zwischengespeichert: im Browser selbst, im Betriebssystem des Browsers oder in mehreren der zur Beantwortung der Anfrage verwendeten DNS-Server. Wird so innerhalb eines gewissen Zeitraumes die selbe Anfrage erneut gestellt, wird anstatt diese weiter zu delegieren, die bereits bekannte Antwort einer früheren Anfrage zurückgegeben. Dieses Zwischenspeichern der Antworten wird als „Caching“ bezeichnet.

Zur Steuerung des Cacheverhaltens wird beim Auflösen der Domain neben der Ziel-IP auch ein Verfallsdatum (TTL, *Time To Live*, engl. für Lebens- oder Gültigkeitsdauer) angegeben, nach welchem die Antwort ihre Gültigkeit verliert und aus dem jeweiligen Cache gelöscht wird. Diese Gültigkeitsdauer lässt sich frei konfigurieren und liegt normalerweise zwischen einigen Sekunden und mehreren Stunden.[26] Wird der TTL-Wert von allen Zwischenspeichern beachtet, lassen sich so die HTTP-Anfragen innerhalb von Sekunden auf einen anderen Server umleiten.

26 Dynamic Network Services Inc, DNS Caching (2010), http://www.dyndns.com/support/kb/dns_caching.html

In der Praxis ist es jedoch so, dass nicht alle Caches kleinere TTL-Werte respektieren und diese länger beibehalten.

- Die DNS-Server im Internet halten sich größtenteils an den TTL-Wert, da dieser zum Standard gehört. Ein abweichendes Verhalten muss konfiguriert oder über den Quellcode programmiert werden. Einige Internetprovider erzwingen einen Mindest-TTL-Wert (z.B. 60 Minuten) um Datenverkehr, Rechenlast und Kosten zu reduzieren. Da heute ein Vielfaches an Rechenleistung und Bandbreite als vor 10 Jahren zur Verfügung steht, bietet dies aber kaum noch Vorteile. Außerdem können Internetdienste, die sich auf geringe TTL-Werte verlassen, für die eigenen Kunden nur eingeschränkt verfügbar sein.[27]
- Der Linux-Betriebssystemkern selbst cached keine DNS-Abfragen. Stattdessen kann der Linux-Systemdienst NSCD (Name Service Cache Daemon) verwendet werden, welcher die DNS-TTL's seit mindestens 2004 respektiert[28]
- Seit Windows 98 werden auch DNS-Abfragen auf Microsoft-Betriebssystemen gecached, welche ebenfalls die TTL-Werte beachten.
- Internet Explorer 4, 5, 7 und 8 cachen alle DNS-Abfragen für 30 Minuten[29] und ignorieren den TTL-Wert gänzlich. Internet Explorer 6 cached keine über DNS aufgelöste IP-Adresse.[30]
- Seit 2004 cachen Firefox und andere Mozilla-Browser die DNS-Einträge für 60 Sekunden unabhängig vom TTL-Wert. Bei früheren Versionen betrug die Cache-Dauer noch 15 Minuten.[31]

Da der Internet-Explorer in Version 7 und 8 mit 30 Minuten DNS-Cache-Zeit mit 24% noch stark verbreitet ist[32], bedeutet dies, dass Failover-Lösungen, welche auf DNS

27 JH Software ApS, DNS Caching and Simple Failover (2006) Seite 2 http://www.simplefailover.com/outbox/dns-caching.pdf
28 Depper, Ulrich, nscd and DNS TTL (2007)http://udrepper.livejournal.com/16362.html
29 Microsoft, How Internet Explorer uses the cache for DNS host entries (2009) http://support.microsoft.com/default.aspx?scid=KB;en-us;263558
30 JH Software ApS, DNS Caching and Simple Failover (2006) Seite 3 http://www.simplefailover.com/outbox/dns-caching.pdf
31 JH Software ApS, DNS Caching and Simple Failover (2006) Seite 3 http://www.simplefailover.com/outbox/dns-caching.pdf
32 Refsnes Data, Browser Statistics (2010), http://www.w3schools.com/browsers/browsers_stats.asp

basieren, mindestens für ein Viertel der potentiellen Besucher nur unzureichend funktionieren können. Der Browser-Cache betrifft aber nur die Besucher, die bereits die Seite vor der DNS-Änderung aufgerufen haben. Neue Besucher erhalten nach der DNS-Umstellung die richtige IP, sofern kein anderer Cache dies verhindert. Es ist jedoch auch möglich, das ein Browser nach einem fehlgeschlagenen Verbindungsversuch unabhängig vom eigenen DNS-Cache erneut eine Anfrage durchführt.

Allgemein gilt das Failover mittels DNS als schlechte Lösung, da der Failover je nach Client innerhalb von mehr als einer Minute bis hin zu mehreren Stunden erfolgen kann.

5.2.2 DNS-Lastverteilung

Es ist möglich, für eine Domain mehrere IP-Adressen anzugeben. Wird diese Domain dann aufgelöst, ist das Ergebnis eine Liste von IP-Adressen, welche bei jeder Anfrage eine andere Anordnung haben. Es existiert kein Standard dafür, welche IP-Adresse aus dieser Liste vom Client verwendet werden soll. Die meisten Clients verwenden einfach die erste IP-Adresse aus der Liste. Seltener ist es der Fall, dass der Client versucht den geographisch nächsten Server zu wählen.

```
;; QUESTION SECTION:
;google.de.                IN      A

;; ANSWER SECTION:
google.de.          300     IN      A       74.125.39.99
google.de.          300     IN      A       74.125.39.104
google.de.          300     IN      A       74.125.39.106
google.de.          300     IN      A       74.125.39.105
google.de.          300     IN      A       74.125.39.103
google.de.          300     IN      A       74.125.39.147
```

Abbildung 7: Mehrere IP-Adressen pro Domain

Ausschnitt des Outputs des Befehls „dig google.de“, welcher 6 Ressourceneinträge des Typs A mit einer TTL von 5 Minuten (300 Sekunden) zurück gibt.

Quelle: eigener Linux-Befehl

Da die IP-Adressen bei jeder Anfrage anders geordnet sind, kann eine gewisse Verteilung der Anfragen und somit eine Lastverteilung realisiert werden. Dies lässt sich aber, im Vergleich zu einem HTTP-Lastverteiler nicht genau steuern, da die meisten Clients für folgende Anfragen immer dieselbe IP verwenden. Ist ein Server mit zu vielen Benutzern überladen, ist es unmöglich, einen Teil der bereits zugreifenden Nutzer mittels DNS auf einen anderen Server umzuleiten. Auch muss dafür Sorge getragen werden, dass nicht mehr verfügbare Server aus dem Adresspool entfernt werden. Im Falle eines Ausfalls besteht hier wieder das Problem, dass die Adressliste an mehreren Stellen

zwischengespeichert wird, und sich somit eine veraltete Liste mit einer nicht mehr erreichbaren Server-Adresse in Umlauf befindet.

Die Lastverteilung per DNS (engl. *Round robin DNS*) bietet sich in Szenarien an, bei denen sich mehrere Server in verschiedenen Rechenzentren befinden. In diesem Fall kann der Ausfall eines gesamten Rechenzentrums mit oben genannten Einschränkungen kompensiert werden. Auch ist diese Art der Lastverteilung dem Failover mit einer einzigen IP vorzuziehen: Wenn bei der Einzel-IP-Failoverlösung der primäre Server ausfällt, ist der Service für alle Clients nicht mehr erreichbar bis diese die neue IP erhalten. Wenn zwei Server mit Lastverteilung mittels DNS zum Einsatz kommen, trifft dies theoretisch nur auf ~50% der Clients zu. Umso mehr Server dabei vorhanden sind um so geringer wird der Anteil an Clients, die vom Ausfall eines einzelnen Servers betroffen sind. Zusätzlich bietet dies den Vorteil, das es auch mit höheren TTL's funktioniert und einige Clients (speziell Browser) auch die alternativen IP-Adressen anfragen und so in kürzerer Zeit wieder die Verbindung mit einem funktionierenden Server aufnehmen können.[33]

Auf Grund der oben genannten Nachteile sollte dies nicht als primäre Lösung zur Erhöhung der Verfügbarkeit eingesetzt werden. Als reine Lastverteilungslösung bei geographisch getrennten Serverstandorten, welche selbst hochverfügbar gehalten werden, bietet sich jedoch die Lastverteilung per DNS an.

5.2.3 Migrierbare IP-Adressen in einem Subnetz

Sind mehrere Server in einem Subnetz vorhanden und besitzen damit dasselbe Routing, können die verfügbaren IP-Adressen zwischen diesen Servern beliebig ausgetauscht werden. Dies bietet die Basis für die Live-Migration von virtuellen Maschinen wie in Kapitel 4.2.2 „Hochverfügbarkeit mit Live-Migration“ beschrieben. Stehen beispielsweise für zwei Server in einem eigenen Subnetz mindestens zwei IP-Adressen zur Verfügung, kann im Falle eines Serverausfalls der verbleibende Server die IP-Adresse des Ausgefallenen übernehmen. So werden mit minimaler Verzögerung auf der IP neu eingehende Anfragen bearbeitet. Vor der Migration vorhandene Netzwerkverbindungen zum ausgefallenen Server werden getrennt, da der

33 NetWidget, Inc., Failover Strategies – Experiment (2007), http://www.netwidget.net/books/apress/dns/info/failover-experiment.html

verbleibende Server von diesen keine Kenntnis besitzt und sie zurückweist, sobald entsprechende Pakete bei ihm eingehen.

Diese Methode eignet sich gut um den Ausfall von Serverhardware zu kompensieren. Es besteht aber die Gefahr, dass aufgrund von Problemen im Rechenzentrum, wie einem lokalen Stromausfall oder dem Ausfall eines einzelnen Switches, gleich alle Server nicht mehr erreichbar sind. Da sich die Server in physikalischer Nähe befinden müssen, können sie auch von lokalen Problemen gleichermaßen betroffen sein.

5.2.4 Failover-IP

Bei einer Failover-IP handelt es sich um eine spezielle IP-Adresse, deren Routing zwischen mehreren Servern und Netzen umgeschaltet wird. Dadurch müssen sich die betreffenden Server nicht im selben Subnetz, wohl aber im selben Rechenzentrum befinden. Nicht alle Rechenzentren bieten dieses Feature an, das sollte bei der Wahl des Anbieters beachtet werden. Üblicherweise erfolgt die Umschaltung der Failover-IP auf einen anderen Server über ein Webinterface oder eine entsprechende API, welche bei einem automatischen Failover angesprochen werden muss. Eine Failover-IP ist eine günstige Möglichkeit, Hochverfügbarkeit zu erreichen, da das Ändern des Routings der Failover-IP nahezu sofort erfolgt. Der Vorteil gegenüber eines eigenen Subnetzes ist, dass sich die betreffenden Server in verschiedenen Sektionen des Rechenzentrums befinden können und somit weniger anfällig für lokal auftretende Probleme sind.

5.2.5 TCP/HTTP-Lastverteiler

Bei einem Lastverteiler auf TCP/HTTP-Ebene handelt es sich um eine Software, welche sich auf einem Server zwischen den aufrufenden Clients und mehreren Zielservern befindet. Gegenüber den Clients agiert der Lastverteiler als Serverdienst, welcher die Anfragen entgegen nimmt und an die Zielserver weiterleitet. Für die Zielserver erscheint der Lastverteiler als Client. Er kann neben der einfachen Weiterleitung von TCP-Verbindungen auch die übertragenen Daten manipulieren. Beispielsweise können bei HTTP-Anfragen die Ziel-URLs von öffentlichen zu internen umgeschrieben oder Header mit den originalen Client-IPs hinzugefügt werden. Auch kann der Lastverteiler einen HTTP-Cookie an den Client herausgeben, um so bei

späteren Aufrufen diesen wieder an denselben Server zu leiten.

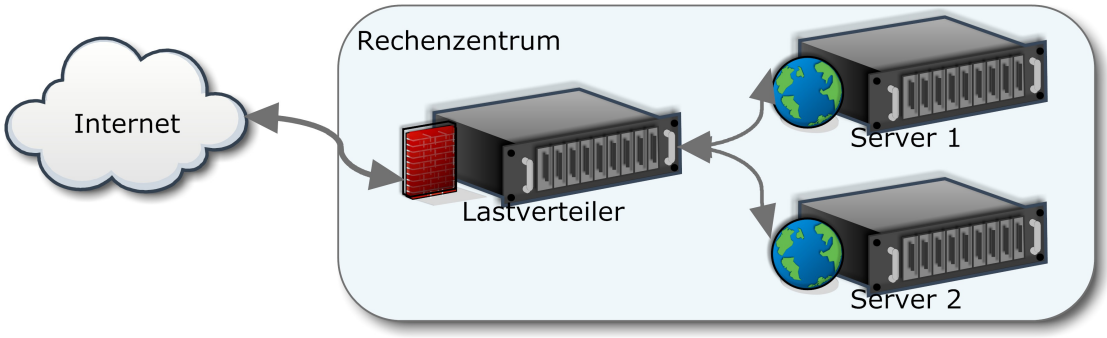

Abbildung 8: Der Lastverteiler verteilt die Zugriffe auf mehrere Server

Quelle: eigene Abbildung

Da ein einzelner lastverteilender Server einen *Single Point of Failure* darstellen würde, muss dieser selbst hochverfügbar gestaltet werden, beispielsweise durch die Verwendung von zwei Lastverteilern mit einer Failover-IP oder einer virtuellen Maschine mit Live-Migration. Letzteres hätte den Vorteil, das aktive TCP-Verbindungen nicht unterbrochen werden.

Im Vergleich zur Lastverteilung per DNS liefert ein solcher Lastverteiler auf TCP-Ebene ein hohes Maß an Kontrolle, da durch intelligente Algorithmen die Vermittlung der Aufrufe gesteuert werden kann. Dies ermöglicht es, einzelne, stark ausgelastete Server zu schonen und wenig ausgelastete zu verwenden. Auch kann ein HTTP-Lastverteiler die Erreichbarkeit der Server prüfen und bei Problemen im Millisekundenbereich reagieren. Zusätzlich bietet ein HTTP-Lastverteiler ein zusätzliches Maß an Sicherheit, da er sowohl unabhängig Aufrufe protokollieren, als auch schädlichen Datenverkehr blockieren kann. Die Leistungsfähigkeit und die verfügbaren Features hängen von der verwendeten Software ab.

6 E-Mail-Server

Es besteht kein Zweifel, dass die E-Mail eine der wichtigsten Kommunikationsmedien der letzten 10 Jahre ist. Gerade im geschäftlichen Bereich ist es wichtig, dass die Zustellung von E-Mails zuverlässig funktioniert. Denn eine verlorengegangene oder als

unzustellbar zum Sender zurückgekehrte E-Mail bedeutet potentiell einen Umsatzverlust – speziell wenn Kunden per E-Mail Bestellungen auslösen oder Geschäftspartner Termine anfordern, kann eine ausbleibende Antwort das Geschäftsverhältnis stören.

6.1 Zustellung von E-Mails mit Backup-Mailserver

Der E-Mail-Standard[34] sieht vor, dass die Zustellung von E-Mails über mehrere Wege möglich ist. Wenn eine E-Mail beispielsweise an „kontakt@helmundwalter.de“ zugestellt werden soll, wird der so genannte „MX“-Eintrag (engl. ***M**ail E**x**changer*) der Domain „helmundwalter.de“ abgefragt. So eine Abfrage kann keine, eine oder mehrere Domains mit möglicherweise unterschiedlichen Prioritäten als Ergebnis liefern, an die eine Zustellung der E-Mail versucht werden kann. Der E-Mail-Standard schreibt vor, dass die Zustellung über mindestens zwei, nach Möglichkeit mehr, der MX-Einträge versucht werden muss.[35]

```
[root@sys ~]dig mx helmundwalter.de
; <<>> DiG 9.6.1 <<>> mx helmundwalter.de
...
;; QUESTION SECTION:
;helmundwalter.de.          IN      MX

;; ANSWER SECTION:
helmundwalter.de.     3600    IN      MX      5 mail1.helmundwalter.de.
helmundwalter.de.     3600    IN      MX      10 mail2.helmundwalter.de.
...
```

Abbildung 9: Gekürzte Ausgabe einer MX-Abfrage mittels "dig"-Befehl unter Linux

3600 entspricht dem TTL von einer Stunde.

Die Zahlen 5 und 10 sind die Prioritäten der MX-Einträge, wobei kleinere Zahlen höher sind.

Quelle: eigene Aufstellung

Dies ermöglicht statt eines einzigen E-Mail-Servers mehrere anzugeben, an die eine E-Mail zugestellt werden kann, wodurch Redundanz entsteht. Üblicherweise betreibt

34 Klensin, J., RFC 2821 – Simple Mail Transfer Protocol (2001), Kapitel 5, http://www.faqs.org/rfcs/rfc2821.html

35 Klensin, J., RFC 2821 – Simple Mail Transfer Protocol (2001), Kapitel 5, http://www.faqs.org/rfcs/rfc2821.html

man neben dem primären E-Mail-Server, welcher die E-Mails in den Postfächern der Nutzer ablegt, einen sogenannten „*Backup-MX*“ oder *E-Mail-Relay-Server*. Dieser zweite Server kann im Problemfall E-Mails entgegennehmen und sie nach Wiederverfügbarkeit des Hauptservers an diesen zustellen. Dies hat den Vorteil, dass die E-Mails erst einmal entgegengenommen werden und der Sender keine Fehlermeldung erhält. Der Nachteil bei diesem Setup ist, dass die E-Mails von den Empfängern nicht abgerufen werden können, wenn der primäre Server nicht verfügbar ist. Somit ist dies zwar eine ausreichende, aber keine ausgezeichnete Lösung.

6.2 Mailserver mit gemeinsamen Speicher

Um die E-Mails trotz Ausfall eines Servers noch verfügbar zu haben, liegt es nahe, die E-Mails statt auf der lokalen Festplatte eines Servers auf einem gemeinsamen Speicher abzulegen. Sowohl der verwendete Postfix-E-Mailserver, als auch der Courier-IMAP/POP3-Server verarbeiten die E-Mails im sogenannten „Maildir++“-Format. Maildir++ ist eine Erweiterung zu Maildir, welches Unterverzeichnisse und Quotas erlaubt. Dabei handelt es sich um ein recht einfaches Speicherformat, bei dem alle E-Mails direkt im Dateisystem mit einzigartigen, beispielsweise auf der Empfangszeit basierenden, Namen gespeichert werden. Dieses Format wurde dafür entworfen, den konfliktfreien Zugriff mehrerer Dienste auf die E-Mails zu gewährleisten. Eine Verwendung in Kombination mit einem Netzwerkspeichergerät und mehreren gleichzeitig arbeitenden E-Mailservern war Bestandteil der Überlegungen des Maildir-Erfinders Daniel J. Bernstein.[36] Daher eignet sich dieses Format bestens, um eine hochverfügbare E-Mail-Lösung zu erstellen.

Um alle E-Mail-Dienste hochverfügbar zu gestalten, werden die E-Mails auf den gemeinsamen Speicher gelegt. Auf Server 1 und 2 werden der Postfix-Mailserver und der Courier-IMAP/POP3-Server unter Verwendung des gemeinsamen Maildirs gestartet. Auf diese Art und Weise können beide Server E-Mails empfangen und konfliktfrei ins Maildir zustellen. Außerdem können die E-Mails von beiden Servern mittels IMAP und POP3 abgerufen werden.

Die Zustellung kann über mehrere MX-Einträge in der Domain, möglicherweise mit gleicher Priorität, an die Server verteilt werden. Die abrufenden Clients müssen

36 Bernstein, Daniel, Using maildir format (1995), http://cr.yp.to/proto/maildir.html

genauso wie die HTTP-Abfragen[37] an einen funktionierenden Server umgeleitet werden. Eine Verteilung der Anfragen auf die vorhandenen Server ist auch hier möglich. Da bei dem Abrufen der E-Mails eine Ausfallzeit von bis zu einer Stunde vertretbar ist, wäre eine DNS-basierende Lösung durchaus ausreichend, wenn auch nicht optimal.

7 MySQL-Datenbankserver – Hochverfügbarkeit und Lastverteilung

Ziel dieses Kapitels ist es, eine produktiv einsetzbare Lösung zu finden, um die MySQL-Datenbank, das Kernstück der meisten Webanwendungen, hochverfügbar und skalierbar zu machen. Für die Hochverfügbarkeit allein, ohne Skalierbarkeit, gibt es viele gute Ansätze. Neben den schon genannten virtuellen Maschinen beispielsweise auch die Lösung aus Kapitel 4.1 „Einfaches applikationsbasierendes Failover mit gemeinsamen Speicher“. Einfaches MySQL-Failover lässt sich auch mit Hilfe von DRBD (**D**istributed **R**eplicated **B**lock **D**evice) realisieren[38]. Die Zielstellung jedoch ist, nicht nur eine hohe Verfügbarkeit, sondern auch eine horizontale Skalierbarkeit, etwa durch Lastverteilung zu erreichen. In diesem Kapitel werden verschiedene Möglichkeiten, einen MySQL-Server skalierbar zu machen vorgestellt, praktisch getestet und bewertet. Das Testen der einzelnen Möglichkeiten ist notwendig, da dadurch erst Probleme offenbart werden, die sonst in einer rein theoretischen Betrachtung keine Beachtung finden würden.

Betrachtet man MySQL unter dem Aspekt der Hochverfügbarkeit und der Lastverteilung und lässt reine Failover-Lösungen außen vor, so gibt es zwei grundlegende Ansätze[39]: zum einen MySQL-Cluster, zum anderen MySQL-Replikation. Worum es sich dabei genau handelt wird in den entsprechenden Kapiteln geklärt.

37 Siehe dazu Kapitel 5.2 „TCP/HTTP Lastverteilung / Anfragenumleitung“
38 Oracle Corporation, MySQL-Referenzhandbuch, 14.1. Using MySQL with DRBD (2010), http://dev.mysql.com/doc/refman/5.1/en/ha-drbd.html
39 Oracle Corporation, MySQL-Referenzhandbuch, Chapter 14. High Availability and Scalability, http://dev.mysql.com/doc/refman/5.1/en/ha-overview.html

7.1 Testsysteme

Beim Experimentieren mit verschiedenen MySQL-Setups entstand die Notwendigkeit, einen Weg zu schaffen, die Leistungsfähigkeit und Anwendbarkeit der Lösungen zu testen. Zum Testen der Kompatibilität und Leistungsfähigkeit der Lösung wird eine Kopie des Webshops verwendet, für welchen diese Lösung später unter anderem gedacht ist. Wenn dieser Shop im Testsetup gut funktioniert, lässt dies die Vermutung zu, dass er auch im Produktiveinsatz die Erwartungen erfüllen wird. Der Shop gehört mit 134346 Artikeln in 2554 Kategorien zu den größten Shops, die von dem Unternehmen betrieben werden.

Als Testsysteme werden zwei virtuelle Maschinen verwendet, die, bis auf die notwendige Konfiguration, identisch sind. Als Virtualisierungstechnik wird QEMU[40] eingesetzt. Jede virtuelle Maschine besitzt 1,5 GB Arbeitsspeicher, keinen Swap-Speicher und eine 2,5 GHz CPU. Beide virtuelle Maschinen sind mit einem Gigabit-LAN verbunden, befinden sich auf verschiedenen Host-Systemen und führen ein 32-Bit ArchLinux aus. Es ist zu erwarten, dass die virtuellen Maschinen deutlich langsamer sind als die Server mit besserer Hardware, auf denen die Lösung später zum Einsatz kommen soll. Da die Bedingungen bei allen Tests gleich sind, lassen sich diese dennoch miteinander vergleichen und sollten Rückschlüsse auf die Leistungsunterschiede der einzelnen Lösungen zulassen.

Die MySQL-Server-Konfiguration wurde, sofern nicht anders angegeben, nicht modifiziert. Einzige Anpassung ist die Deaktivierung des Query-Caches. Der Query-Cache speichert Ergebnisse zu einmal ausgeführten Anfragen zwischen und würde so das Testergebnis möglicherweise verfälschen. Als gemeinsamer Speicher wird vorerst eine Samba-Freigabe verwendet, welche mittels Cifs in das Dateisystem beider virtueller Maschinen eingebunden ist.

Die Ausführungszeiten der einzelnen Tests variieren, was unter anderem damit zu erklären ist, dass sowohl das Betriebssystem, als auch der MySQL-Prozess Daten im Arbeitsspeicher halten und die Testsysteme neben dem MySQL-Dienst auch andere Prozesse ausführen, welche CPU und Arbeitsspeicher verwenden. Um dennoch ein aussagekräftiges Ergebnis zu erhalten, werden die einzelnen Tests 5 mal wiederholt

40 Bellard, Fabrice, About QEMU (2010), http://wiki.qemu.org/Main_Page

und der Durchschnitt der Ausführungszeiten gebildet.

Die Tests werden von einem dritten System im selben Netzwerk ausgeführt, so dass die durch die Testprogramme entstehende CPU-Last das Ergebnis nicht verfälscht.

7.2 Die Tests im Detail

Die aufgeführten Tests sollen Aufschluss darüber geben, wie die veränderten Bedingungen der einzelnen Server-Setups sich auf die Betriebsgeschwindigkeit der gehosteten Projekte auswirken. Die Tests sind keine allgemeingültigen Aussagen über die Leistungsfähigkeit der einzelnen Lösungen, da diese aufgrund vieler Faktoren stark variieren können. Sie lassen auch keine direkten Schlüsse darüber zu, wie schnell die produktiven Systeme die geforderten Aufgaben verarbeiten werden.

Faktoren, die die Leistungsfähigkeit beeinflussen, sind unter anderem:

- Geschwindigkeit der CPU
- Anzahl CPU´s und Kerne
- 32- oder 64-Bit, Betriebssystem und MySQL-Version
- Konfigurationseinstellungen, ebenfalls Betriebssystem und MySQL
- Lesegeschwindigkeit von Dateien (Raidcontroller, Festplatten, Mainboard)

Bei den Tests geht es darum, die verschiedenen Lösungen miteinander zu vergleichen, wobei die oben genannten Ressourcen nach Möglichkeit konstant gehalten werden.

7.2.1 Test 1: Produktlisten, Suchfunktion, Produktdetailseite

Dieser Test ist dem Besuch eines Kunden im Webshop nachempfunden. Dabei werden größtenteils lesende Abfragen auf die Datenbank ausgeführt.

Das Testskript ruft erst die Startseite auf, verwendet anschließend zweimal die Suchfunktion, wobei einmal ein einzelner und einmal 479 Artikel gefunden werden. Von den 479 Artikeln werden widerum 200 angezeigt. Anschließend werden zwei Produktlisten in verschiedenen Kategorien aufgerufen, wobei eine Kategorie nur einen

und die andere 309 Artikel enhält, von denen ebenfalls 200 angezeigt werden. Zum Schluss folgen zwei Aufrufe von Produktdetailseiten.

Die folgende Tabelle gibt Aufschluss über die einzelnen Seitenaufrufe und die daraus resultierenden SQL-Abfragen, sowie der Ausführungszeit eines einzelnen Durchganges mit einem einfachen MySQL-Server der Version 5.1.44 in einem der virtuellen Testsysteme.

Seite	***SELECT***	***UPDATE***	***INSERT***	***Gesamt***	***Zeit in s***	***Beschreibung***
Startseite	69	1	1	71	0,16	Startseite des Shops
Suche 1	837	1	1	839	14,05	Zeigt 200 von 479 Ergebnissen
Suche 2	41	1	1	43	14,48	Zeigt 1 Ergebnis
Produkt-Listing 1	1106	1	1	1108	3,49	Zeigt 200 von 309 Ergebnissen
Produkt-Listing 2	90	1	1	92	3,79	Zeigt 1 Ergebnis
Produkt-Detail-Seite 1	95	2	1	98	1,13	
Produkt-Detail-Seite 2	57	2	1	60	0,11	
Gesamt	2295	9	7	2311	37,21	

Tabelle 4: Seitenaufrufe und daraus resultierende SQL-Abfragen

Quelle: eigene Messungen

Wie zu sehen ist, führt eine Suchanfrage oder eine Produktauflistung mit vielen Artikeln zu enorm vielen SQL-Abfragen, was die Folge von schlecht optimierten Code ist. Für jedes gefundene Produkt werden jeweils noch weitere Abfragen durchgeführt, die beispielsweise spezielle Produktattribute, Sonderpreise und Rabatte für jedes Produkt einzeln auslesen. Dabei handelt es sich um das normale Vorgehen des osCommerce-Shops und leider auch vieler anderer Webanwendungen. Dies ließe sich deutlich optimieren, was aber nicht Thema dieser Thesis ist. Der Fairness halber muss allerdings gesagt werden, dass der Shop normalerweise nur 20 Artikel pro Seite anzeigt und es somit nicht sehr ins Gewicht fallen würde. Mit 200 Artikeln pro Seite (Kundenwunsch) liefert der schlechte Code viele Queries, die für Testzwecke jedoch nicht schaden. Bei Testumgebungen, in denen zwei Server vorhanden sind, wird dieser Test nur auf einem ausgeführt.

7.2.2 Test 2: Eine komplexe Abfrage

Eine Abfrage mit Unterabfrage, „COUNT(*)“, „GROUP_CONCAT2, „JOIN“ über 5 Tabellen, „LIKE“, „GROUP BY“, „ORDER BY“ und „LIMIT“. Die Abfrage wird im System verwendet um alle Artikel und deren EANs zu finden, welche bei einer externen Schnittstelle aktualisiert werden sollen. Dabei liefert das Query pro EAN immer nur den günstigsten und verfügbarsten Artikel. Derart komplexe Abfragen werden oft von Importen und Exporten verwendet. Zwar sind diese nicht zeitkritisch, da sie im Hintergrund laufen, jedoch erlaubt eine schnelle Ausführung der Importe bei Bedarf ein höheres Wiederholungsintervall. So können die Artikeldaten beispielsweise nicht nur alle 20, sondern alle 10 Minuten aktualisiert werden.

Das Query liefert 38443 Ergebnisse. Da es nur um die Ausführung des Queries, nicht um das Laden und Anzeigen der Ergebnisse geht, wird dieses mit LIMIT 1 ausgeführt. Das bewirkt, dass der SQL-Server dennoch das gesamte Ergebnis berechnet, aber nur die erste Zeile davon an den Client sendet. Die Ausführung der Anfrage erfolgt mit Hilfe des MySQL-Kommandozeilen-Clients. Der SQL-Code der Anfrage befindet sich im Anhang.

7.2.3 Test 3: Seitenaufrufe mehrerer Benutzer

Bei diesem Test werden neben Test 1 noch 3 weitere Varianten desselben Testprogrammes parallel gestartet, mit dem Unterschied, dass diese die Seiten in anderer Reihenfolge aufrufen. Dies soll zeigen, wie sich der MySQL-Server bei mehreren gleichzeitigen Abfragen verhält. Dies ist besonders wichtig, um die Skalierbarkeit der Lösung bezüglich lesender Abfragen bewerten zu können. Das Ergebnis dieses Tests ist der Durchschnitt der 4 Teilausführungszeiten.

Wenn zwei Server an dem Testsetup beteiligt sind, werden jeweils 2 Prozesse auf einem Server ausgeführt.

7.2.4 Test 4: Schreibende Operationen

Die Tests 1 bis 3 bestehen zum größten Teil aus lesenden Datenbankoperationen, führen also intern den SELECT-Befehl in verschiedenen Variationen aus. Wie die genauere Betrachtung der einzelnen Lösungen zeigen wird, ist es möglich, dass nur

lesende Abfragen skalierbar werden, schreibende aber nach wie vor von allen Knoten ausgeführt werden müssen.

Um die Auswirkungen der Lösung auf schreibende Operationen sichtbar zu machen, ist es nötig, dies explizit zu erproben. Der Ablauf des Tests sieht wie folgt aus:

1. Test 4 verwendet eine leere Datenbanktabelle als Ausgangspunkt.
2. Einfügen von 50.000 Datensätzen mit INSERT-Befehlen.
3. Überprüfen des Ergebnisses mit Hilfe des SELECT-Befehls aus Abbildung 10. Dieser liefert die Ausgabe „YES YES YES YES“, sofern die Anzahl der Datensätze und die Summe der Spalten „price“, „dist“ und „quantity“ den erwarteten Werten entsprechen.

```
SELECT
IF(count(*)=50000,
    "YES","NO") as count,
IF(sum(price)=30510522.10,
    "YES","NO") as mark1,
IF(sum(dist)=983050349255,
    "YES","NO") as mark2,
IF(sum(quantity)=1254427,
    "YES","NO") as mark3
FROM `benchmark1`
```

Abbildung 10: Select zur Überprüfung der Ergebnisse

Quelle: eigenes Query

4. Überschreiben der Datensätze mit Nullwerten durch einen UPDATE-Befehl, welcher keine WHERE-Klausel besitzt, und somit alle Datensätze erfasst.
5. Wiederherstellen der Datensätze unter Verwendung von 50.000 UPDATE-Befehlen.
6. Wiederholtes Überprüfen des Ergebnisses mit dem SELECT-Befehl aus Abbildung 10.

Dieser Test ist an ein Verfügbarkeitsimportskript aus der Praxis angelehnt, welches neue Verfügbarkeitsdatensätze zur späteren Auswertung in eine Tabelle importiert. Das reale Skript startet allerdings nicht mit einer leeren Tabelle, sondern mit einer vollen, in der sich teilweise veraltete Daten befinden.

7.2.5 Test 5: Mehrfache Ausführung von Test 4

Um zu prüfen wie gut sich schreibende Abfragen skalieren lassen, werden bei diesem Test vier Instanzen von Test 4 parallel gestartet. Dabei arbeitet jede Instanz in einer eigenen Datenbanktabelle, so dass es nicht zu Kollisionen kommen kann. Wie auch

bei Test 3 wird der Durchschnitt der Teilausführungszeiten aller Instanzen gebildet.

Wenn zwei Server an dem Testsetup beteiligt sind, werden jeweils zwei Instanzen auf einem Server ausgeführt, sofern dies möglich ist.

7.2.6 Test 6: Einfache Abfragen

Test 1, welcher als Basis für Test 3 dient, ruft Seiten des Shopsystems auf, welche eine große Zahl an Datenbankanfragen generieren. Darunter sind teilweise komplexe Abfragen, die sich über mehrere Tabellen erstrecken und Textstücke im Volltext suchen. Im Gegensatz zu diesen komplexen Abfragen soll Test 6 Aufschluss darüber geben, wie sich der SQL-Server mit einfachen Abfragen verhält, welche in der Praxis häufig vorkommen. Mit einfachen Abfragen sind solche gemeint, die nur nach dem Primärschlüssel suchen und sich auf eine einzige Tabelle beziehen. Dieser Test führt 135.000 Abfragen nach dem Muster „SELECT products_model FROM products WHERE products_id = '1424223'“ durch, wobei die Artikelnummer „products_id“ bei jeder Abfrage anders ist. Auf diese Art wird jeder Artikel aus der osCommerce-Produkttabelle einzeln anhand seines Primärschlüssels abgefragt.

7.3 Standard MySQL-Server

Bei diesem Setup handelt es sich um die Ausgangssituation: ein einfacher MySQL-Server ohne spezielle Konfiguration. Die Standardkonfiguration wird getestet, um die Ergebnisse für den Vergleich mit allen weiteren Tests heranziehen zu können. Um möglichst gleiche Bedingungen zu schaffen, wird an dieser Stelle die Binärdistribution des 32-Bit-MySQL-Servers der Version 5.1.44 von Mysql.com verwendet. Die Alternative wäre gewesen, den MySQL-Server über den Paketmanager der Linux-Distribution zu installieren. Dies hätte aber den Nachteil gehabt, dass die Version der Distribution mit dem GCC-Kompiler, allerdings die Variante von Mysql.com mit Intel CC kompiliert wurde und sie schon deswegen unterschiedlich schnell sein könnten. Auch ist die MySQL-Versionsnummer der Linux-Distribution höher, was ebenfalls der Vergleichbarkeit abträglich wäre.

7.3.1 Installation

Im Vergleich zur Komplexität und Funktionalität eines MySQL-Servers ist die

Installation und Einrichtung eines solchen ausgesprochen simpel. MySQL steht für Linux, Windows, Solaris und Mac als Binärpaket auf der Website von MySQL in 32- und 64-Bit Versionen bereit.[41] Zu Testzwecken wird bei diesem und folgenden Tests immer die 32-Bit Linux-Variante verwendet.

Nach dem Herunterladen muss das MySQL-Paket entpackt werden. Das daraus resultierende Verzeichnis wird nach /usr/local/mysql-default verschoben.

- `tar xfvz mysql-5.1.44-linux-i686-glibc23.tar.gz`
- `mv mysql-5.1.44-linux-i686-glibc23 /usr/local/mysql-default`

Dann muss ein neuer Nutzer und eine Gruppe angelegt werden, unter der der MySQL-Server unprivilegiert ausgeführt werden kann.

- `groupadd mysql`
- `useradd -g mysql mysql`

Um den Server einfacher starten zu können, wird das mitgelieferte Startskript an die für die Linux-Distribution richtige Stelle kopiert:

- `cp /usr/local/mysql-default/support-files/mysql.server /etc/rc.d/mysqld-default`

Damit das Startskript die MySQL-Distribution und das MySQL-Datenverzeichnis findet, müssen die zwei entsprechenden Variablen im Skript editiert werden:

- `basedir=/usr/local/mysql-default`
- `datadir=/var/lib/mysql`

7.4 Konfiguration, Start und Einspielen der Shopdatenbank

Der Standardpfad, unter welchem der MySQL-Server die MySQL-Konfigurationsdatei erwartet ist /etc/my.cnf. Die Konfiguration wird für diesen Test nur benötigt, um den standardmäßig aktivierten Query-Cache abzuschalten. Nachdem diese Datei wie in Abbildung 11 erstellt ist, kann der MySQL-Server mit folgendem Befehl gestartet

41 http://dev.mysql.com/downloads/mysql/

werden:

```
[mysqld]
query_cache_type = 0
```

Abbildung 11: Minimale MySQL-Konfigurationsdatei /etc/my.cnf

Quelle: eigene Konfigurationsdatei

- `/etc/rc.d/mysqld-default start`

Ist dies geschehen, ist es an der Zeit, die Shopdatenbank in den MySQL-Server einzuspielen. Da es sich um eine SQL-Datei, also eine Datei mit Befehlen zum Erstellen und Befüllen der Datenbanktabellen handelt, bietet es sich an, diese mit dem „mysql“-Kommandozeilenclient auszuführen. Das Zeichen '<' leitet dabei die Datei „shop.sql“ an die Standardeingabe des Programmes, so dass die darin enthaltenen Befehle sofort ausgeführt werden.

- `/usr/local/mysql-default/bin/mysql < /root/shop.sql`

Nachdem dies abgeschlossen ist, steht dem Shop die Datenbank zur Verwendung bereit. Es folgen nun die Messungen der einzelnen Tests.

7.4.1 Testergebnisse

	Lauf 1 in s	***Lauf 2 in s***	***Lauf 3 in s***	***Lauf 4 in s***	***Lauf 5 in s***	***Durchsch. Lauf 1-5***
Test 1	37,65	37,19	37,51	37,45	37,1	37,38
Test 2	11,97	12,67	11,81	11,62	11,8	11,97
Test 3	143,26	143,41	142,9	143,97	143,76	143,46
Test 4	31,71	31,55	30,61	31,67	30,78	31,26
Test 5	122,69	123,67	122,79	122,54	123,82	123,1
Test 6	34,66	34,58	34,27	33,6	34,16	34,25
Summe						381,43

Tabelle 5: Testergebnisse mit der MySQL-Standard-Installation

Quelle: eigene Berechnungen und Messungen

Beim Vergleich von Test 1 mit Test 3 fällt auf, dass Test 3 nur um 284% langsamer als bei Test 1 ist. Da es sich bei Test 3 um die vierfache Ausführung von Test 1 handelt liegt die Vermutung nahe, dass dieser auch 4 mal so lange dauert, also 300% langsamer wäre. Der Unterschied von 16% lässt sich damit erklären, dass bei parallel ausgeführten Abfragen die Hardware-Ressourcen effizienter genutzt werden können. Beispielsweise kann der MySQL-Server schon das Ergebnis einer Abfrage berechnen,

während er bei einer Anderen im selben Moment Daten von der Festplatte liest. Bei Test 4 und 5 tritt dieser Effekt weniger stark auf, da der Unterschied zum Erwartungswert von 125,04[42] Sekunden bei diesen Test nur rund zwei Sekunden beträgt.

7.5 MySQL-Cluster

MySQL-Cluster ist eine Technologie, mit der sich MySQL verteilt im Netzwerk betreiben lässt, was hohe Performance, Skalierbarkeit und eine Ausfallsicherheit von 99,999% verspricht[43]. Kernstück von MySQL-Cluster ist die NDB-Storage-Engine. NDB steht für **N**etzwerk **D**aten**b**ank (engl. *network database*). Dabei handelt es sich um eine verteilte Speicher-Engine für MySQL, welche 2003 durch die Übernahme der Firma Alzato, einem Unternehmen der Firma Ericsson und deren Produkt „NDB Cluster" durch MySQL AB, Teil des MySQL-Servers wurde. Ein MySQL-Server mit NDB-Unterstützung und allen Diensten und Tools, welche zum Betrieb des Clusters notwendig sind, werden in einem Paket als MySQL-Cluster angeboten[44]. Auch hier existiert eine Open-Source- und eine kommerzielle Variante, welche unter anderem zusätzliche grafische Tools zur Verwaltung des Clusters beinhaltet und erlaubt, im Gegensatz zur Open-Source-Variante, neue Knoten ohne einen Neustart des Clusters hinzuzufügen. Die komerzielle Variante trägt die Bezeichnung „Carrier Grade Edition" oder kurz CGE. Die Open-Source-Variante wird wie bei dem MySQL-Server als „*Generally Available*" kurz GA, als Community- oder auch, nach der Lizenz, als GPL-Version bezeichnet. Zur Zeit des Schreibens ist die Version 7.1.3 von MySQL-Cluster aktuell, welche zu Testzwecken im Weiteren verwendet wird. Die Versionsnummerierung ist unabhängig von der MySQL-Serverversion.

7.5.1 Funktionsweise

Um einen MySQL-Cluster zu betreiben werden drei unterschiedliche Computer-prozesse benötigt. Dabei handelt es sich um Serverprogramme, welche alle auf einem oder auch auf unterschiedlichen Rechnern betrieben werden können. Sie werden im Weiteren clustertypisch als Knoten bezeichnet. Bei den drei Knotenarten handelt es

42 Die vierfache Dauer von Test 4
43 http://www.mysql.com/products/database/cluster/
44 http://www.mysql.com/downloads/cluster/

sich um Datenknoten (Prozessname „ndbd“), Managementknoten (ndb_mgmd) und SQL-Knoten (mysqld). MySQL-Cluster basiert auf der „*Shared-Nothing-Architecture*“, was bedeutet, dass jeder Knoten unabhängig ist und über eine eigene CPU, Festplatte und Arbeitsspeicher verfügt. Dadurch, dass bei MySQL-Cluster alle der genannten Knotentypen mehrmals vorhanden sein können, besitzt es keinen *Single Point of Failure*, also kein System, bei dessen alleinigen Ausfall der Datenbank-Cluster funktionsunfähig werden würde. Es wird auch kein gemeinsamer Speicher benötigt, da MySQL-Cluster das Verteilen und Speichern der Daten selbstständig verwaltet.

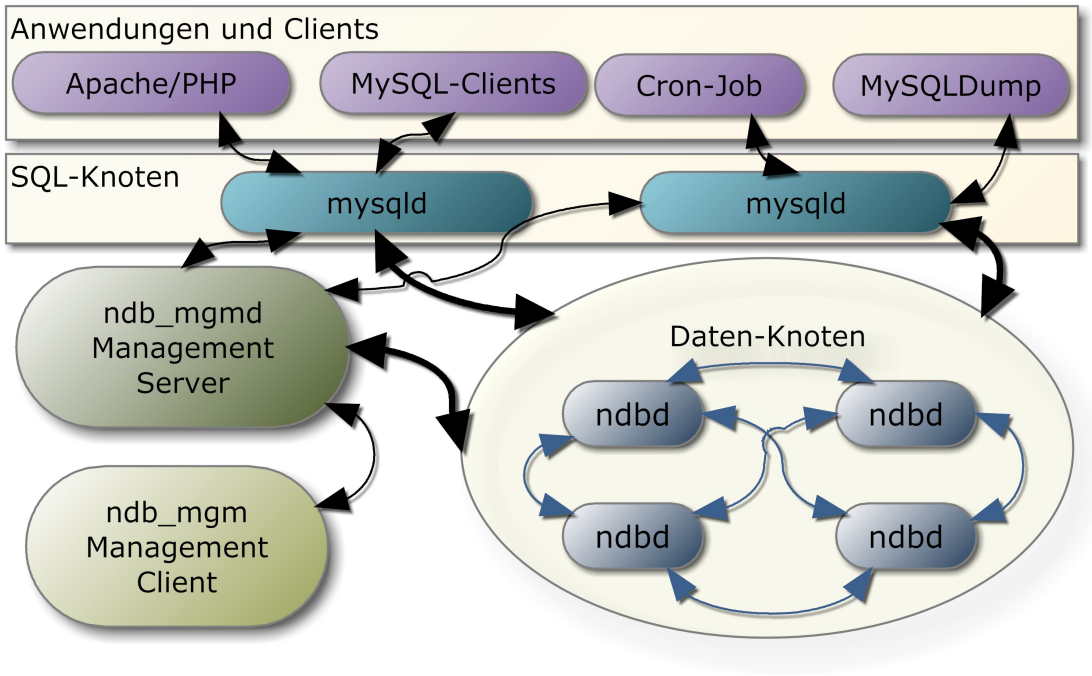

Abbildung 12: Aufbau und Funktionsweise des MySQL-Clusters.

Quelle: eigene Abbildung, angelehnt an http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-overview.html

7.5.1.1 Datenknoten

Wie in der Einführung zu MySQL Cluster erwähnt, ist das Kernstück von MySQL-Cluster die Speicher-Engine NDB. Diese benötigt zum Funktionieren einen oder mehrere Datenknoten, auf denen die Daten gespeichert und anschließend wieder

ausgelesen werden können. Die Daten werden dabei zwischen zwei oder mehr Datenknoten repliziert, so dass bei Ausfall eines Datenknotens diese immernoch vorhanden und zugänglich sind. Synonym wird für diesen Knotentyp auch die Bezeichnung NDB-Knoten verwendet.

Die Daten werden bei MySQL-Cluster standardmäßig im Arbeitsspeicher der Datenknoten gehalten und periodisch zur Sicherung auf die Festplatte geschrieben. Außerdem werden alle Änderungen seit der Sicherung als MySQL-Binärlog gespeichert, so dass nach einem Vollausfall des Clusters dennoch der letzte Zustand wiederhergestellt werden kann, sofern dieser schon in das Binärlog geschrieben wurde. Das Schreiben des Binärlogs erfolgt asynchron, so dass möglicherweise dennoch Transaktionen verloren gehen können.

Das Halten der Daten im Arbeitsspeicher hat natürlich den Vorteil, dass diese von da aus sehr schnell und effizient abfragbar sind, aber auch den Nachteil, dass Arbeitsspeicher nicht in den Größenordnungen zur Verfügung steht, wie es bei Festplattenspeicher der Fall ist. Seit MySQL-Cluster Version 6 gibt es die Möglichkeit, einzelne Spalten dauerhaft auf die Festplatte auszulagern, wobei Spalten mit Indizes nach wie vor im Arbeitsspeicher gehalten werden müssen.[45]

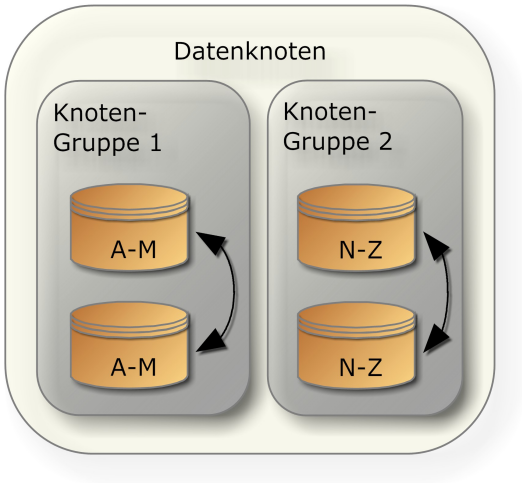

Abbildung 13: Aufteilung der Daten auf Datenknoten

Quelle: eigene Abbildung

7.5.1.1.1 Knotengruppen

Um trotz der intensiven Arbeitsspeichernutzung dennoch große Mengen an Daten verwalten zu können, werden die vorhandenen Datenknoten zu sogenannten Knotengruppen aufgeteilt. Innerhalb dieser Knotengruppen, welche aus mindestens einem Knoten bestehen müssen, werden die Daten repliziert, wobei bei mehr als einer Knotengruppe jede Knotengruppe nur einen Teil der Gesamtmenge der

45 Oracle Corporation, MySQL-Referenzhandbuch, 17.5.10. MySQL Cluster Disk Data Tables (2010) http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-disk-data.html

Daten enthält. Ein Ausfall eines Knotens einer Knotengruppe hat also keinen Datenverlust zur Folge, sofern dieser Knoten nicht der letzte seiner Knotengruppe war.

Die Verwendung von Knotengruppen bietet den Vorteil, dass der MySQL-Cluster auch in schreibintensiveren Umgebungen dennoch skalierbar bleibt, da Änderungen am Datenbestand nur von einzelnen Knotengruppen und nicht vom gesamtem Cluster durchgeführt werden müssen. Dennoch eignet sich MySQL-Cluster am besten für Anwendungen, die hauptsächlich lesende und nur wenige schreibende Anfragen generieren.

7.5.1.1.2 Schlüssel-Partitionierung

Die Tabellenstruktur und die Inhalte der Tabellen werden in sogenannten Partitionen gespeichert. Die Anzahl der Partitionen im Cluster entspricht der Anzahl der Knoten, da jeder Knoten eine Partition beinhaltet.

Die Aufteilung der Daten auf die Knotengruppen erfolgt anhand einer Funktion des Primärschlüssels jeder NDB-Tabelle. Wenn eine Tabelle laut Tabellendefinition keinen Primärschlüssel besitzt, so erstellt NDB automatisch einen künstlichen, impliziten Primärschlüssel, welcher in der Tabellendefinition nicht sichtbar ist.

Um Redundanz herbeizuführen, werden die Partitionen zwischen allen Knoten einer Knotengruppe repliziert. Die Kopien einer Partition werden in der MySQL-Terminologie sekundäre Replikate oder einfach Replikate genannt. Das Original wird primäres Replikat genannt.

Die „*NoOfReplicas*“-Cluster-Konfigurations-Variable legt die Anzahl dieser Kopien und damit auch die Größe der Knotengruppen fest. Da die Partitionen innerhalb der Knotengruppe von *n* Knoten repliziert werden, beinhaltet jeder Knoten seine eigene primäre Partition und *n-1* Kopien der anderen. Auch muss die Gesamtzahl der beteiligten Knoten ein Vielfaches von „*NoOfReplicas*“ sein, da alle Knotengruppen gleich groß sein müssen. Der benötigte Arbeitsspeicher aller beteiligten Knoten ergibt sich somit aus der Gesamtmenge der zu speichernden Daten multipliziert mit der Anzahl der Replikate.

Wenn ein Cluster vier Knoten besitzt, und „*NoOfReplicas*“ gleich zwei ist, so entstehen zwei Knotengruppen mit jeweils zwei Knoten. Das erste primäre Replikat

befindet sich auf dem ersten Knoten der ersten Gruppe, von welcher gleichzeitig eine Kopie auf dem zweiten Knoten eingerichtet wird. Das zweite primäre Replikat wird auf dem ersten Knoten der zweiten Gruppe erstellt, die dritte auf dem zweiten Knoten der ersten und die vierte auf dem zweiten Knoten der zweiten mit jeweils einer Kopie auf dem anderen Knoten derselben Gruppe.

Die Tabellen werden standardmäßig in Teile mit jeweis 32 Kilobyte Größe aufgeteilt und gleichmäßig auf die verfügbaren Partitionen verteilt. Ist eine Tabelle kleiner als 32 Kilobyte, so wird diese nur auf einer Partition gespeichert.

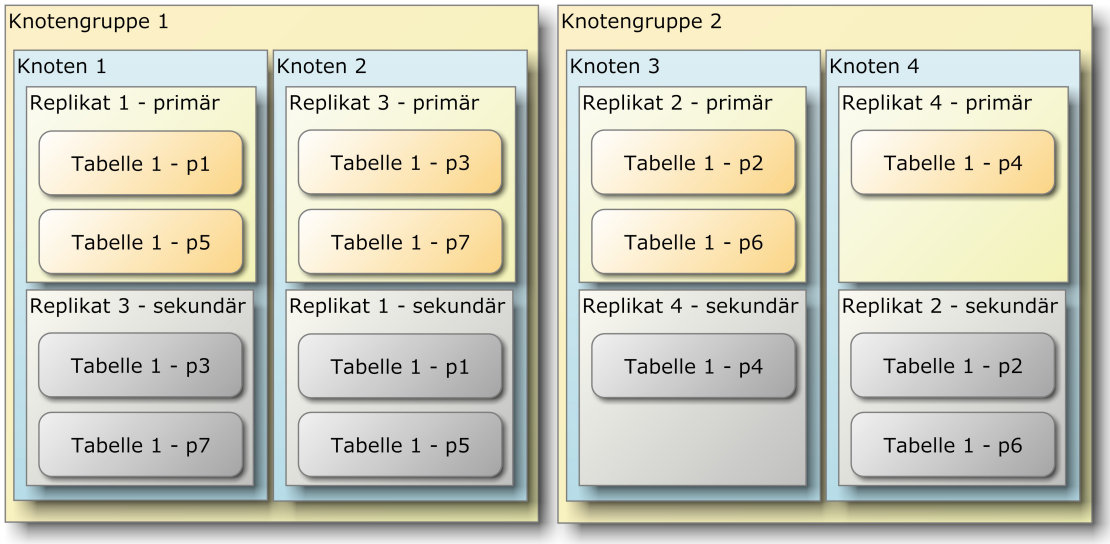

Abbildung 14: Partitionen und Replikate bei MySQL-Cluster

Quelle: eigene Abbildung

Abbildung 14 zeigt die Aufteilung einer Tabelle in sieben Teile (p1-7) auf vier Partitionen. Weitere Tabellen werden nach demselben Prinzip gleichmäßig verteilt. Die Aufteilung der Nutzdaten auf die Partitionen bietet aufgrund der hohen Abstraktion viel Flexibilität, welche beim Hinzufügen von neuen Knotengruppen und Knoten als auch für die Wiederherstellung von permanent ausgefallenen Knoten benötigt wird.

Fällt beispielsweise Knoten 2 aus, so übernimmt Knoten 1 vollständig die Kontrolle über die lokal replizierten Daten von Knoten 2, kann diese also lesen und bearbeiten. Wenn Knoten 2 später wieder ins Cluster aufgenommen wird, so werden die von Knoten 1 protokollierten und durchgeführten Änderungen auf Knoten 2 übertragen.

Nach erfolgreicher Resynchronisierung kann Knoten 2 dann wieder seine normale Tätigkeit im Cluster aufnehmen.

Die sekundären Replikate sind weder les- noch schreibbar, solange das primäre Replikat noch aktiv ist. Es wäre günstig, wenn sowohl primäre- als auch sekundäre Replikate lesbar wären, da so in einem Cluster mit einer Knotengruppe beispielsweise jeder Knoten jede Abfrage beantworten könnte, ohne die anderen Knoten zu kontaktieren. Dies könnte ein Feature für zukünftige MySQL-Cluster-Versionen sein, um die Performance zu verbessern.

7.5.1.2 SQL-Knoten

Bei SQL-Knoten handelt es sich um eine, mit NDB-Unterstützung kompilierte, Version des MySQL-Servers. Neben dem NDBCluster- oder NDB-Tabellentyp stehen auch alle anderen MySQL-eigenen Tabellentypen wie InnoDB und MyISAM zur Verfügung, so dass man diese dennoch für Aufgaben verwenden kann bei denen Ausfallsicherheit und Skalierbarkeit keine Rolle spielt. Der SQL-Knoten kommuniziert direkt mit den Datenknoten, nimmt SQL-Abfragen von den Clients entgegen und liefert die Ergebnisse an diese zurück. Da der SQL-Knoten eine Variante des MySQL-Servers ist, lautet der Prozessname ebenfalls „mysqld“ und wird auch als „MySQL-Knoten“ bezeichnet.

7.5.1.3 Managementknoten

Der Managementknoten ist für die Konfiguration des Clusters und die Verwaltung der Knoten verantwortlich. Initialisiert man ein MySQL-Cluster, so ist der Managementknoten der erste, der gestartet werden muss. Dieser liest die Konfigurationsdatei ein und wartet auf eingehende Verbindungen, über die er relevante Konfigurationsparameter an die anderen Knoten weitergibt, die Verbindung zu ihm aufnehmen. Diese benötigen als einzige Konfiguration die Adresse, unter welcher der Managementknoten erreichbar ist. Außerdem ist der Managementknoten in der Lage, andere Knoten auf entfernten Systemen neuzustarten oder das gesamte Cluster herunterzufahren. Auch protokolliert der Managementknoten Ereignisse innerhalb des Clusters wie Knotenausfälle oder knapp werdenden Arbeitsspeicher.

Der Managementserver ist notwendig, um das Cluster zu starten oder bei einem

Ausfall einzelne Daten- oder SQL-Knoten wieder in das Cluster einzugliedern. Für den ganz normalen Clusterbetrieb nach dem Start ist der Managementserver nicht nötig. Dennoch ist es empfehlenswert, dass der Managementserver stetig verfügbar ist, da Ausfälle meist nicht geplant erfolgen.

7.5.1.4 Die Rolle des Arbitrators

Bei mehr als zwei Datenknoten muss ein Datenknoten neben seinen normalen Aufgaben auch die Rolle des Masters (engl. für Meister) übernehmen. Der Master enthält für die Arbeitsfähigkeit des Clusters essenzielle Informationen über den Zustand des Clusters. Jeder Datenknoten des Clusters kann die Rolle des Masters bei Bedarf übernehmen. Unter Umständen ist ein Arbitrator, ein Schiedsrichter notwendig, um zu entscheiden, welcher Datenknoten nun Master werden soll. Standardgemäß ist der Managementknoten der Arbitrator, aber auch SQL-Knoten können so konfiguriert werden, dass sie diese Rolle übernehmen. Ein automatischer Wechsel des Arbitrators findet nicht statt.

Bei der Initialisierung des Clusters ist immer der erste Datenknoten, der den Managementserver, bzw. den Arbitrator, kontaktiert automatisch Master.

Fällt der Master bei laufenden Betrieb aus, so gibt es zwei Möglichkeiten, wie ein neuer Master gewählt wird:

1. Die absolute Mehrheit der Datenknoten ist noch online und kann miteinander kommunizieren, dann wählen diese einen neuen Master.
2. Wenn weniger oder exakt die Hälfte der Datenknoten noch online ist, wird der Arbitrator kontaktiert, welcher dann entscheidet, wer der neue Master wird.

Dieses Vorgehen ist notwendig, um das sogenannte *Split-Brain*-Szenario (engl. für geteiltes Gehirn) zu vermeiden. Dabei handelt es sich nicht um ein MySQL-Cluster-spezifisches Problem, sondern eines, welches generell bei Computerclustern auftreten kann. Dabei fällt die Kommunikation zwischen zwei oder mehr Teilen des Clusters aus, was dazu führt, dass jeder Teil davon ausgeht, die jeweils anderen Teile wären vollständig ausgefallen und sie müssten jetzt alleinig alle Aufgaben übernehmen. Dies ist bei reinen Lesezugriffen relativ unproblematisch. Werden die getrennten Teile des Clusters jedoch unabhängig voneinander beschrieben, ist ein anschließendes

Zusammenführen der Clusterteile und deren im Konflikt stehenden Datenbeständen ein problematisches und oft auch nur manuell zu lösendes Unterfangen. Für ein Cluster ist ein *Split-Brain* so gesehen oft schlimmer als ein vollständiger Cluster-Ausfall, da bei letzteren nach der Wiederherstellung wenigstens alle Daten noch konsistent und wiederspruchsfrei sind.

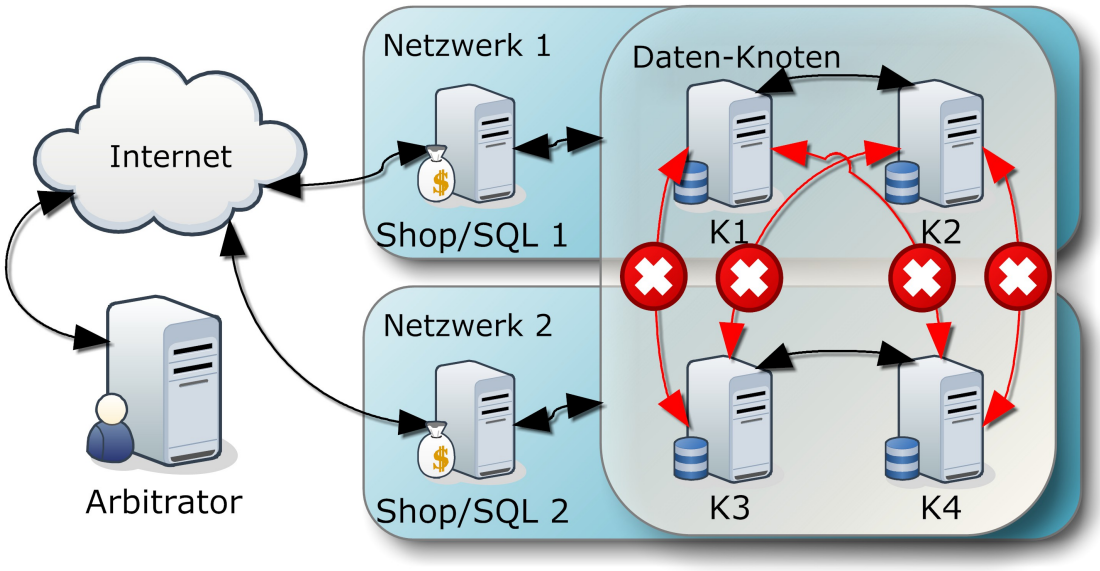

Abbildung 15: Split-Brain-Szenario
Quelle: eigene Abbildung

Das Beispiel aus Abbildung 15 soll das Problem verdeutlichen. Die Ausgangssituation ist ein MySQL-Cluster mit 4 Datenknoten (K1-K4). K1 und K2 befinden sich in Netzwerk 1, die anderen Knoten in Netzwerk 2. Beide Netzwerke könnten sich innerhalb eines Rechenzentrums befinden, welches intern zwischen diesen Netzen routet und beide mit dem Internet verbindet. Im Ausgangszustand kann jeder Knoten mit jedem anderen kommunizieren und K1 ist der Datenknoten-Master. In jedem Netzwerk befindet sich außerdem noch ein SQL-Knoten und ein Webserver, über den Kunden den Shop besuchen und Bestellungen auslösen können.

Fällt nun das Routing zwischen diesen beiden Netzen aus, können die Datenknoten K1 und K2 nicht mehr mit K3 und K4 kommunizieren und umgekehrt. Die Internetverbindung ist aber immer noch intakt, so dass Kunden sowohl über SQL1 und SQL2 auf den Webshop zugreifen können. K1, der bisherige Master, und K2 müssen

nun davon ausgehen, da sie K3 und K4 nicht mehr erreichen, dass diese offline, also defekt sind und setzen den Betrieb fort. K3 und K4 haben widerum keinen Master mehr, welcher benötigt wird, um wieder funktionsfähig zu werden. Sollten jetzt K3 und K4 die anderen Knoten für „tot“ erklären, den Knoten K3 zum neuen Master wählen und den Betrieb fortsetzen, wären aus einem Cluster nun zwei entstanden. Da beide Cluster voll funktionsfähig sind, aber nichts von der Existenz des anderen Clusters wissen, spricht man nun von „*Split-Brain*“. Kunden könnten sowohl auf SQL1 als auch auf SQL2 Bestellungen auslösen, so dass ab da beispielsweise Bestellnummern und Kundennummern doppelt vergeben werden.

Dieses Problem lässt sich mit Hilfe eines Arbitrators lösen, welcher sich am Besten außerhalb des Rechenzentrums der Datenknoten befindet. Fällt nun die Verbindung zwischen Netzwerk 1 und Netzwerk 2 aus, nehmen die Datenknoten Kontakt zum Arbitrator auf, sobald sie bemerken, dass ein Teil der Knoten nicht erreichbar ist. Der Teil, der zuerst den Arbitrator erreicht wird instruiert, die Cluster-Tätigkeiten fortzusetzen. Der andere Teil, der möglicherweise auch aufgrund von Internetverbindungsproblemen den Arbitrator verspätet erreicht, wird aufgefordert, sich herunterzufahren. Die heruntergefahrenen Knoten können dann, wenn das Netzwerkproblem gelöst ist, wieder in den Cluster aufgenommen und neu synchronisiert werden.

Bei dem Entwurf eines MySQL-Clusters müssen solche Szenarien bedacht werden, um den Cluster möglichst ausfallsicher zu gestalten. Befände sich der Arbitrator im Netzwerk 1, so könnten die Knoten aus Netzwerk 2 nicht übernehmen, wenn Netzwerk 1 ausfällt, womit der gesamte Cluster offline wäre. Wenn Knotengruppen gebildet werden, die einen Teil der Daten untereinander replizieren, dann sollten K1 und K3 zur ersten und K2 und K4 zur zweiten Gruppe zusammengefasst werden, so dass K1 und K2 genauso wie K3 und K4 eine Kopie des gesamten Datenbestandes enthalten, falls ein Netzwerk ausfällt.

7.5.2 Aufbau des Testsetups

Zur Realisierung eines MySQL-Clusters stehen zwei Server zur Verfügung, was ausreichend ist, um das Cluster grundsätzlich redundant aufzubauen. Für echte Redundanz wird, wie im vorherigen Kapitel beschrieben, ein Arbitrator außerhalb des

Clusters benötigt. Der Arbitrator benötigt kaum Hardwareressourcen und kann daher bei Bedarf auch problemlos auf einem günstig angemieteten virtuellen Server oder einem einfachen Shell-Account betrieben werden. Für dieses Testsetup spielt das aber noch keine Rolle. Die Verteilung der MySQL-Knoten auf die beiden virtuellen Testserver wird nach Abbildung 16 realisiert.

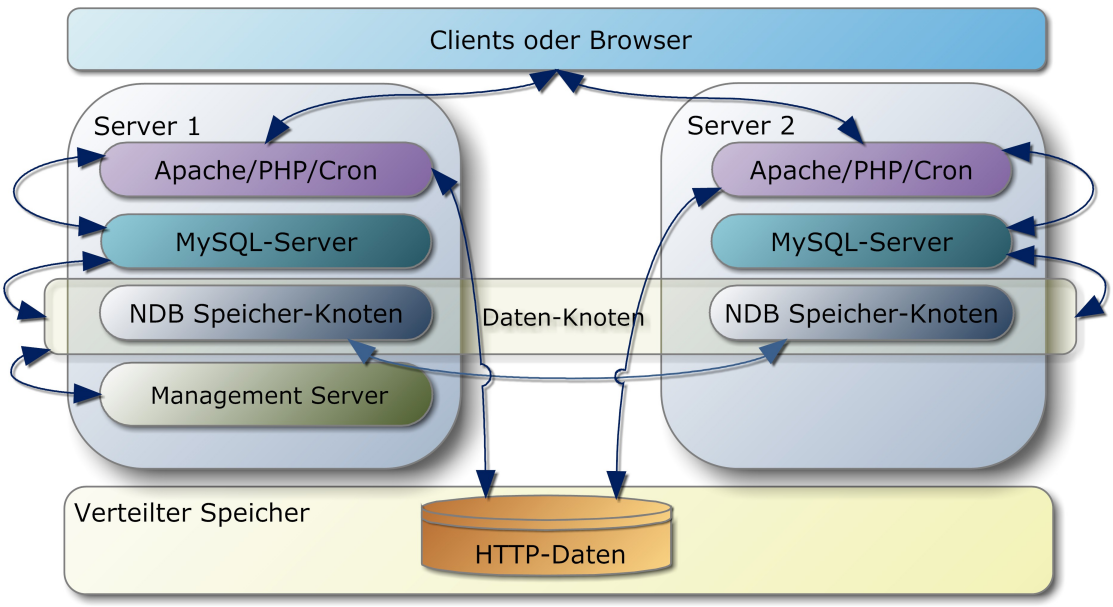

Abbildung 16: Aufbau eines MySQL-Clusters mit zwei Servern.
Der hellere blaue Pfeil entspricht der clusterinternen Replikation.
Quelle: eigene Abbildung

7.5.3 Einrichtung

Die Einrichtung von MySQL-Cluster ist, wie die des Standard-MySQL-Servers, ebenfalls sehr einfach gehalten. Wie der Standard-MySQL-Server steht auch MySQL-Cluster als Binärpaket zum Download auf mysql.com bereit. Die Installation wird anhand von MySQL-Cluster Version 7.1.3, der MySQL 5.1.44 zugrunde liegt, beschrieben.

7.5.3.1 Installation

Nach dem Herunterladen[46] wird die Binärdistribution entpackt, anschließend wird das entstandene Verzeichnis nach /usr/local/mysql verschoben.

- `tar xfvz mysql-cluster-gpl-7.1.3-linux-i686-glibc23.tar.gz`
- `mv mysql-cluster-gpl-7.1.3-linux-i686-glibc23 /usr/local/mysql`

Falls noch nicht geschehen, müssen Nutzer und Gruppe „mysql“ angelegt werden.

- `groupadd mysql`
- `useradd -g mysql mysql`

Anschließend muss das MySQL-Startskript kopiert werden:

- `cp /usr/local/mysql/support-files/mysql.server /etc/rc.d/mysqld`

Da /usr/local/mysql der Standardpfad für die MySQL-Binärdistribution ist, muss dieser im Startskript nicht explizit angegeben werden.

7.5.3.2 Konfiguration

Um den Aufbau des Clusters zu beschreiben, muss die Cluster-Konfigurationsdatei definiert werden. Die Konfiguration gliedert sich in Sektionen, welche mit eckigen Klammern und dem Sektionsnamen eingeleitet werden. Die Sektion „`[NDBD DEFAULT ]`“ legt für alle Knoten gültige Parameter fest, welche bei der Knotenkonfiguration nicht nochmals angegeben werden müssen. Der Parameter „`NoOfReplicas`“ gibt an, wieviele Kopien einer Tabelle im Cluster gehalten werden sollen und legt somit gleichzeitig die Größe der Knotengruppen fest.[47] Dieser Wert kann nur in der „`[NDBD DEFAULT]`“-Sektion angegeben werden, da dieser für den gesamten Cluster einheitlich sein muss.

46 http://dev.mysql.com/downloads/cluster/#downloads
47 Oracle Corporation, MySQL-Referenzhandbuch, 17.3.2.6. Defining MySQL Cluster Data Nodes (2010), http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-ndbd-definition.html

MySQL-Cluster enthält standardmäßig alle Daten im Arbeitsspeicher. Die Parameter „DataMemory“ und „IndexMemory“ legen fest, wieviel Speicher für die Daten und die Indizes zur Verfügung stehen. Die Standardwerte sind 80 MB für Daten und 10 MB für Indizes. Dies reicht nicht aus um die gesamte Shopdatenbank (ca. 400 MB) aufzunehmen, weswegen diese Werte erhöht werden müssen. Mit „DataDir“ wird festgelegt, wo die Nutzdaten, also die Sicherungskopien und Binärlogs auf den jeweiligen Datenknoten gespeichert werden. Für die „[NDB_MGMD DEFAULT]“-Sektion legt der Parameter „DataDir“ fest, wo der binäre Konfigurationscache des Clusters abgelegt wird.

```
[NDBD DEFAULT]
NoOfReplicas=2
DataMemory=480M
IndexMemory=80M
DataDir= /usr/local/mysql/data
[MYSQLD DEFAULT]
[NDB_MGMD DEFAULT]
Datadir=/usr/local/mysql/mgm
[NDB_MGMD]
HostName=192.168.0.210
Id=1
[NDBD]
Id=3
HostName=192.168.0.210
[NDBD]
Id=4
HostName=192.168.0.220
[MYSQLD]
HostName=192.168.0.210
[MYSQLD]
HostName=192.168.0.220
```

Abbildung 17: Datei /usr/local/mysql/config.ini

Quelle: eigene Konfigurationsdatei

In den folgenden Sektionen werden ein Managementknoten, zwei Datenknoten und zwei SQL-Knoten definiert. Die Knoten innerhalb eines Clusters benötigen eine eindeutige ID, mit der sie identifiziert werden können. Außerdem wird mit „HostName“ die IP der Knoten festgelegt, welche vom Managementknoten beim Verbinden überprüft wird, so dass nur Knoten mit der richtigen IP dem Cluster beitreten können.

Schlussendlich muss noch die Konfigurationsdatei /etc/my.cnf angelegt werden, welche den SQL- und Datenknoten mitteilt, unter welcher Adresse der Managementserver zu erreichen ist. Die Sektion „[mysqld]“ betrifft den SQL-Knoten. In dieser wird, wie in Kapitel 7.1 „Testsysteme“ gefordert, mit „query_cache_type = 0“ der Query-Cache deaktiviert. Die Direktive „ndbcluster“ teilt dem MySQL-Server mit, dass er die Kommunikation mit dem NDB-Cluster aufnehmen soll. Wie der Clustermanager zu erreichen ist, wird

```
[mysqld]
query_cache_type = 0
ndbcluster
ndb-connectstring=192.168.0.210

[mysql_cluster]
ndb-connectstring=192.168.0.210
```

Abbildung 18: MySQL-Cluster Knotenkonfiguration /etc/my.cnf

Quelle: eigene Konfigurationsdatei

mit „`ndb-connectstring`“ angegeben. Selbiges gilt auch für die „`[mysql_cluster]`“-Sektion, welche von den NDB-Datenknoten gelesen wird. Diese Konfigurationsdatei muss folglich auch auf Server 2 vorhanden sein, damit der dortige NDB-Knoten und der SQL-Knoten die Verbindung aufnehmen können.

7.5.3.3 Starten des Clusters

Nachdem die Installation und Konfiguration abgeschlossen ist, müssen die einzelnen Knoten nur noch gestartet werden. Dabei muss zuerst der Managementserver auf Server 1 initialisiert werden, da dieser zum Starten und Verbinden aller anderen Knoten benötigt wird.

- `/usr/local/mysql/bin/ndb_mgmd -f /usr/local/mysql/config.ini`

Sollte mehr als ein Managementknoten in der Konfiguration angegeben sein, so müssen diese auch gestartet werden. Anschließend wird der NDB-Knoten und der SQL-Knoten gestartet, jeweis auf beiden Servern.

- /usr/local/mysql/bin/ndbd
- /etc/rc.d/mysqld start

Mit der Cluster-Management-Konsole, welche auf der Kommandozeile mit dem Befehl „ndb_mgm“ gestartet wird, kann der Zustand des Clusters überprüft werden.

Wie in Abbildung 19 zu sehen, besitzt die Management-Konsole einen Eingabebereich, in den Befehle zur Steuerung und Abfrage des Clusters eingegeben werden können. Der Befehl zum Anzeigen des Clusterstatus lautet „SHOW“, wobei Groß- oder Kleinschreibung keine Rolle spielt. Auch ein abschließendes Semikolon, wie bei SQL-Abfragen üblich, ist optional. Die Ausgabe zeigt, gruppiert nach Knotentyp, die verbundenen Knoten des Clusters inklusive Knotennummer, IP-Adresse und MySQL-Version an. Bei den Datenknoten wird ebenfalls die Knotengruppenangehörigkeit (NodeGroup 0) angezeigt. Das Schlüsselwort „Master“ kennzeichnet den aktuellen Master-Datenknoten, in diesem Falle der Datenknoten auf Server 1, da dieser als erstes gestartet wurde.

```
[root@Server1 mysql]# ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> show;
Connected to Management Server at: 192.168.0.210:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=3   @192.168.0.210  (mysql-5.1.44 ndb-7.1.3, Nodegroup: 0, Master)
id=4   @192.168.0.220  (mysql-5.1.44 ndb-7.1.3, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1   @192.168.0.210  (mysql-5.1.44 ndb-7.1.3)

[mysqld(API)]   2 node(s)
id=5   @192.168.0.210  (mysql-5.1.44 ndb-7.1.3)
id=6   @192.168.0.220  (mysql-5.1.44 ndb-7.1.3)

ndb_mgm>
```

Abbildung 19: Die Cluster-Management-Konsole zeigt den Zustand des Clusters, hier vollständig aktiv

Quelle: selbst erzeugter Konsolenlog

7.5.3.4 Einspielen der Shopdatenbank

Nachdem der MySQL-Cluster voll funktionsfähig ist, muss die Datenbank des Shops eingerichtet werden. Im Unterschied zum Standard-MySQL-Server müssen die Shoptabellen aber nicht von Typ MyISAM, sondern vom Typ NDBCluster sein, damit diese den MySQL-Cluster verwenden. Um dies zu erreichen gibt es zwei Ansätze: Die SQL-Datei kann angepasst und importiert werden, so dass die Tabellen gleich als NDBCluster-Tabellen erstellt und befüllt werden. Alternativ können die importierten Tabellen nachträglich zu NDBCluster konvertiert werden.

Das Editieren der SQL-Datei per Hand gestaltet sich aufgrund der Dateigröße und der Menge an zu ändernden Stellen sehr aufwendig. Ein automatisches „Suchen und Ersetzen“ ist nicht 100% zuverlässig, da ein solcher Ausdruck auch andere Vorkommen der Zeichenkette „MyISAM“ ersetzen würde. Auch unter Verwendung eines möglichst akuraten, regulären Ausdruckes lässt sich dies nicht vollkommen ausschließen. Auch ist nicht gesagt, dass alle SQL-Dateien, die importiert werden sollen, dieselbe Formatierung und Syntax haben. Das „Suchen und Ersetzen“ ist keine saubere, zukunftsfähige Lösung, die man auch auf einem Produktivsystem einsetzen könnte, weswegen das nachträgliche Konvertieren der Tabellen bevorzugt wird.

Wie schon bei der Einrichtung des Standard-MySQL-Server-Setups beschrieben, wird

erst einmal die unveränderte SQL-Datei mittels „mysql“-Kommandozeilenprogramm importiert:

- `/usr/local/mysql/bin/mysql < /root/shop.sql`

Die gerade importierten Tabellen sind vom Typ MyISAM und müssen nun mit Hilfe des „ALTER TABLE“-Befehl zu NDBCluster konvertiert werden. Da dies für jede Tabelle separat geschehen muss, bietet es sich an, für die 76 Tabellen des Shops ein kleines Skript wie in Abbildung 20 zu verwenden.

```
<?php
mysql_connect("localhost", "root","");
mysql_select_db("shop");
$q = mysql_query("SHOW TABLES");
while($c = mysql_fetch_array($q))
{
mysql_query("ALTER TABLE `".$c[0]."`
          ENGINE = NDBCLUSTER")
   or var_dump(mysql_error());
}
?>
```

Abbildung 20: PHP-Skript für das Konvertieren aller Tabellen einer Datenbank zum NDBCLUSTER

Quelle: eigener Code

Dieses Skript stellt zuerst eine Verbindung zum Datenbankserver her, wechselt anschließend zur Datenbank „Shop“ und liest mit dem Befehl „SHOW TABLES“ alle Tabellennamen der Shopdatenbank aus. Mittels While-Schleife wird über alle Tabellennamen iteriert, welche jeweils in den Befehl „ALTER TABLE `Tabellen-Name` ENGINE = NDBCLUSTER“ eingesetzt werden. Dieser Befehl wird an den Datenbankserver übermittelt, welcher daraufhin den Tabellentyp zu NDBCluster konvertiert. Es bietet sich an, während dieses Prozesses den Management-Client zu starten, so dass Clusterfehler oder Probleme bei der Konvertierung sofort sichtbar werden. Beispielsweise schlägt die Konvertierung fehl, wenn auf den Datenknoten nicht genügen Arbeitsspeicher zur Verfügung steht, um die neue Tabelle aufzunehmen.

Ist die Konvertierung abgeschlossen, steht die Shopdatenbank nun beiden SQL-Knoten und somit auch beiden Shopinstallationen gemeinsam zur Verfügung und kann nun ausführlich getestet werden.

7.5.4 Testergebnisse

	Lauf 1 in s	***Lauf 2 in s***	***Lauf 3 in s***	***Lauf 4 in s***	***Lauf 5 in s***	***Durchsch. Lauf 1-5***	***Standard-Mysql in s***	***Änderung in %***
Test 1	1784,8	1748,3	1754,1	1752,7	1762,3	1760,44	37,38	4609,58
Test 2	57,46	55,72	54,84	55,51	55,9	55,89	11,97	366,73
Test 3	2562,7	2535,7	2575,7	2543,7	2557,7	2555,1	143,46	1681,05
Test 4	236,62	237,45	237,69	236,94	237,21	237,18	31,26	658,64
Test 5	367,34	365,63	366,66	366,82	366,42	366,57	123,1	197,78
Test 6	82,95	82,84	81,63	81,53	81,55	82,1	34,25	139,68
Summe						5057,28	381,43	1225,86

Tabelle 6: Testergebnisse von MySQL-Cluster

Quelle: eigene Berechnungen und Messungen

Die Testergebnisse zeigen, dass MySQL-Cluster bei den ausgeführten Tests deutlich langsamer ist als die Standardversion. Dennoch lassen sich aus dem Ergebnis interessante Rückschlüsse ziehen. Der Vergleich zwischen Test 1 und Test 3 zeigt, dass durch Parallelisierung eine höhere Gesamtperformance erreicht werden kann. Eine einzelne Instanz von Test 1 benötigt im Schnitt etwa 29 Minuten, die viermalige Ausführung von Test 3 mit ca. 43 Minuten benötigt jedoch nicht einmal doppelt so lange. Bei dem Standard-MySQL-Server ist Test 3 um 283% langsamer als bei Test 1, was dem Erwartungswert von 300% recht nahe kommt. Der Unterschied beider Tests beträgt bei MySQL-Cluster dagegen nur 43,61%. Ähnlich verhält es sich mit dem Verhältnis bei den schreibenden Tests 4 und 5, da die Differenz hier 54,55% beträgt.

Aufgrund des Ergebnisses ergaben sich die folgenden Fragen:

- Warum sind Test 3 und Test 5 trotz einer Vervierfachung der Arbeit nur 43,61% bzw. 54,55% und nicht etwa 100% langsamer?
- Warum ist MySQL-Cluster bei den vorgenommenen Tests so viel langsamer als der Standard-MySQL-Server?

Zur Beantwortung dieser Fragen muss die interne Funtkionsweise von MySQL-Cluster genauer betrachtet werden. Wie in Kapitel 7.5.1.1.2 „Schlüssel-Partitionierung“ beschrieben, werden die Daten im MySQL-Cluster verteilt gespeichert. Dies ist günstig, wenn zu großen Teilen nur Primärschlüsselabfragen ausgeführt werden, aber

sehr ungünstig, wenn zur Beantwortung einer Abfrage mehrere Tabellen benötigt werden, wie es bei JOIN der Fall ist. MySQL-Cluster selbst besitzt z.Z. keine Unterstützung für JOIN. Wird dennoch ein Join auf mehrere NDB-Tabellen ausgeführt, so löst der MySQL-Server diesen mit Hilfe des sehr aufwendigen „*Nested-Loop Join* Algorithmus“ auf.[48] Bei diesem Algorithmus wird jede Zeile der ersten Tabelle, welche die WHERE-Bedingungen erfüllt, in einer Schleife durchlaufen, in welcher widerum eine Schleife für die zweite Tabelle und deren Bedingungen eingebettet ist.

```
/* EXPLAIN */ SELECT count( * )
FROM products p JOIN products_description pd ON
p.products_id = pd.products_id
JOIN products_to_categories ptc ON p.products_id =
ptc.products_id
WHERE
p.products_status =1
AND p.products_availability_id =0
AND p.products_ean IS NOT NULL
AND pd.products_url IS NOT NULL
```

Abbildung 21: Beispiel für einen JOIN mit 3 Tabellen

Quelle: eigener SQL-Befehl, abgeleitet von 7.2.2 „Test 2: Eine komplexe Abfrage“

Der in Abbildung 21 gezeigte SQL-Befehl vereinigt die drei Tabellen „products p“, „products_description pd“ und „products_to_categories p2c“ mit Hilfe von zwei expliziten EQUI-Joins. Wird das auskommentierte „EXPLAIN“ einkommentiert, so erläutert der SQL-Server die Reihenfolge, in welcher die Tabellen gejoint werden: erst „products“, dann „products_to_categories“ und zum Schluss „products_description“. Der gezeigte Befehl braucht bei dem vorgestellten Testsetup 44,68 Sekunden um eine Antwort zu finden. Um diese recht lange Verarbeitungszeit zu erklären, ist es wichtig zu wissen, wie lange eine einzelne Primärschlüsselabfrage benötigt. Die Antwort dazu liefert uns „Test 6: Einfache Abfragen“. Dieser benötigt im Schnitt 82,1 Sekunden um 135.000 einzelne Primärschlüssel-SELECT-Abfragen durchzuführen, was 608 µs pro Abfrage entspricht. Dieser Wert wird als *a* bezeichnet.

Da laut EXPLAIN zuerst die „products“-Tabelle bearbeitet wird, führt der SQL-Server zuerst eine Operation ähnlich dem Befehl aus Abbildung 22 aus, welcher b = 447.522 µs benötigt und r_1 = 38.677 Datensätze betrifft. Anschließend führt der SQL-Server in einer Schleife 38.677 einzelne

```
SELECT count( * )
FROM products p
WHERE p.products_status =1
AND p.products_availability_id =0
AND p.products_ean IS NOT NULL
```

Abbildung 22: Select auf die "products"-Tabelle

Quelle: eigener SQL-Befehl

48 Oracle Corporation, MySQL-Referenzhandbuch,7.3.1.8. Nested-Loop Join Algorithms (2010) http://dev.mysql.com/doc/refman/5.1/en/nested-loop-joins.html

Abfragen auf „products_to_categories“ durch, was insgesamt dem Befehl aus Abbildung 23 entspricht, der 27,6 Sekunden benötigt und r_2 = 38.681 Zeilen als Ergebnis liefert. Die Formel aus Abbildung 24 dient zur Berechnung der Gesamtdauer eines JOIN innerhalb von MySQL-Cluster. Das Ergebnis kann nur eine Annäherung sein, da *a* nur ein Näherungswert ist und in der Praxis auch andere Faktoren die Geschwindigkeit beeinflussen können.

```
SELECT count( * )
FROM products p JOIN
products_to_categories p2c
ON p.products_id =
p2c.products_id
WHERE p.products_status =1
AND p.products_availability_id
=0
AND p.products_ean IS NOT
NULL
```

Abbildung 23: Join der beiden Tabellen "products" und "products_to_categories"

Quelle: eigene Abfrage

Laut dieser Formel berechnet sich die Zeit der Zwischenabfrage aus Abbildung 23, welche real 27,6 Sekunden benötigte, wie folgt:

$$447522\mu s + (38677 * 608\mu s) = 23{,}96\,s$$

$$b + \sum_{x=1}^{n} (r_x * a)$$

Abbildung 24: Formel zur Berechnung der Dauer eines JOIN

Quelle: nach Rechenbeispiel von Jonas Oreland, „MySQL-Cluster and push-down joins (in pursuit of the holy grail)“, Seite 17

Die Zeit der ursprünglichen Abfrage aus Abbildung 21, welche 44,68 Sekunden dauerte, analog dazu:

$$447522\mu s + (38677 * 608\mu s) + (38681 * 608\mu s) = 47{,}48\,s$$

Zusammenfassend führt der SQL-Server $38677 + 38681 = 77358$ Unterabfragen durch, wobei jede Abfrage ca. 608 µs benötigt. Der Standard-SQL-Server benötigt für diese Abfragen nur 34,25 Sekunden, also 253 µs pro Abfrage. Die Differenz der Geschwindigkeit einer einzelnen Abfrage zwischen MySQL-Cluster und einem einzelnen MySQL-Server beträgt also durchschnittlich 355 µs, was in etwa der Netzwerklatenz zwischen den beiden NDB-Knoten entspricht. Setzt man *a* = 355, so ergibt dies 27,91 s potentielle Netzwerklatenz. MySQL-Cluster wartet also etwa 59% der Gesamtzeit auf Antworten des Netzwerkes.

Dies erklärt auch, warum Test 3 und Test 5 trotz einer Vervierfachung der Arbeit nur 43,61% bzw. 54,55% langsamer sind: Das Warten auf die Antworten aus dem Netzwerk wird parallelisiert, bei Vierfacher Ausführung warten letztendlich 4 Instanzen gleichzeitig auf Antworten. Wird Test 1 ausgeführt, so bewegt sich die CPU-Auslastung auf Grund der Netzwerklatenz auf dem SQL-Knoten bei ca. 30% und auf beiden NDB-

Knoten auf ca. 20%, was bedeutet, dass die Hardwareressourcen mit einer einfachen Ausführung der Tests nicht voll ausgelastet werden konnten. Erst durch die Parallelisierung der Anfragen wird die CPU-Leistung der Hardware voll genutzt.

Momentan befindet sich eine Alternative zu den „*Nested-Loop-Joins*“ in Entwicklung, welche allerdings noch nicht für den produktiven Einsatz geeignet ist und auch nur für EQUI-Joins unter bestimmten Bedingungen funktioniert. Diese Methode wird vorläufig als „*Pushed Down Joins*“ (engl. für nach unten gereichte Joins) oder auch SPJ (*Select, Project, Join*) bezeichnet. Dabei reicht der SQL-Server die Aufgabe des Join an die Datenknoten weiter, welche den Join unter sich auflösen, was bei ersten Tests in einigen Fällen eine Geschwindigkeitssteigerung von etwa 2500% bewirken soll.

Der Test der SPJ-Vorschauversion war leider nicht möglich, da das Starten des Clusters immer mit einem Absturz der Software endete.[49]

7.5.5 Einschätzung

MySQL-Cluster ist eine sehr leistungsfähige Software, welche es einfach macht, eine Datenbank verteilt im Netzwerk zu betreiben. Anders als bei MySQL-Replikation verwaltet MySQL-Cluster den Ausfall einzelner Datenknoten und die Wiedereingliederung dieser selbstständig und ohne Neustart des Clusters. Für Anwendungen, welche größtenteils Primärschlüsselanfragen verursachen, eignet sich MySQL-Cluster sehr gut. Speziell wenn eine größere Anzahl an Datenknoten zum Einsatz kommt, ist dies eine zuverlässige und gut skalierbare Lösung.

Da die meisten vom Unternehmen gehosteten Webanwendungen jedoch exzessiven Gebrauch von JOIN´s machen, ist die Antwortzeit von einer einzelnen Abfrage bereits deutlich zu hoch, weswegen die Bearbeitung eines Seitenaufrufes unzumutbar lange dauert. Dies könnte sich mit der Weiterentwicklung von MySQL-Cluster in Zukunft ändern, da die Technik prinzipiell noch viel Raum für Optimierungen lässt. Sollte das Unternehmen in Zukunft eine Webanwendung neu entwickeln, von welcher hohe Skalierbarkeit erwartet wird, so ist es eine gute Möglichkeit, diese von Anfang an für den Betrieb mit MySQL-Cluster zu optimieren. Zur Zeit eignet sich MySQL-Cluster für den vom Unternehmen angestrebten Anwendungszweck aber leider nicht.

49 Helm, Bernd, Ndb crashing at startup (2010), http://bugs.mysql.com/bug.php?id=54310

7.6 MySQL-Replikation

Replikation ist eine Funktion von MySQL, welche es erlaubt, Daten und Tabellen von einem MySQL-Datenbankserver auf einen anderen zu replizieren. Replizieren bedeutet, dass die Daten auf den beteiligten Servern im Cluster redundant gehalten und synchronisiert werden. Dies funktioniert auch zwischen verschiedenen, von MySQL unterstützten Betriebssystemen und Hardware-Architekturen.[50] Dabei fungiert ein MySQL-Server als Master, auf dem alle Schreiboperationen ausgeführt werden. Die geänderten Daten werden dann asynchron auf den oder die Slave-MySQL-Server übertragen, welche ihrerseits die Daten nicht verändern, wohl aber beliebig lesen können. Asynchron bedeutet zum einen, dass der Master bei geänderten Daten nicht auf die Bestätigung der Slaves warten muss und andererseits, dass die Slaves nicht ständig mit dem Master verbunden sein müssen, um zu funktionieren. Dies steht in Kontrast zur synchronen Replikation des MySQL-Clusters, bei welcher die Daten sofort an alle verfügbaren Knoten einer Knotengruppe weiter gegeben werden müssen. Aufgrund der Asynchronität spielt die Netzwerklatenz, anders als bei MySQL-Cluster, keine Rolle bei der Performance. Die Replikation kann sogar über mehrere Tage hinweg verzögert werden. Dies lässt sich nutzen, um sowohl Ausfallsicherheit als auch Skalierbarkeit zu erreichen. Auch zur Datensicherung oder zu Testzwecken lässt sich dies verwenden, da es sich anbietet in diesem Fall mit einer Kopie der Livedaten zu arbeiten.

7.6.1 Replikation im Detail

Alle Änderungen, die auf dem Master stattfinden, werden in einen MySQL-Binärlog geschrieben. In diesem Binärlog werden alle Ereignisse gespeichert, die Daten oder Struktur verändern, so dass ein Slave anhand dieses Binärlogs alle ändernden Ereignisse nachvollziehen kann. Der Binärlog kann in drei verschiedenen Formaten[51] geführt werden: statementbasierend, zeilenbasierend oder gemischt. Bei der statementbasierenden Replikation werden alle ändernden SQL-Statements gespeichert, die dann genauso auf den Slaves ausgeführt werden können. Bei der zeilenbasierenden Replikation werden anstelle des Statements, die aus dem

50 Oracle Corporation, MySQL-Referenzhandbuch, FAQ (2010), http://dev.mysql.com/doc/refman/5.1/en/replication-faq.html#qandaitem-17-4-4-1-13
51 http://dev.mysql.com/doc/refman/5.1/en/replication-formats.html

Statement resultierenden, modifizierten Tabellenzeilen gespeichert, so dass diese Zeilen auf Seite des Slaves einfach überschrieben bzw. eingefügt oder gelöscht werden können. Bei dem gemischten Binärlogformat ändert der MySQL-Server den Log-Modus je nach Art des Ereignisses. Anders als MySQL-Cluster Version 6 und höher, welche standardmäßig zeilenbasierend arbeiten, ist bei der MySQL-Replikatikation die statementbasierende Variante seit MySQL Version 5.1.29 das Standard-Binärlog-Format.

Wenn ein Slave die Verbindung mit dem Master aufnimmt, fragt der Slave eine Kopie des Binärlogs an.[52] Anders als bei MySQL-Cluster sendet der Master also nicht selbstständig die Änderungen an die Slaves, sondern beantwortet nur deren Anfragen nach Änderungen. Der Slave lädt den vom Master gesendeten Binärlog mit den auszuführenden Ereignissen in seinen eigenen, sogenannten „*Relay-Log*“, in dem im zweiten Schritt alle neuen Änderungen ausgeführt werden. Der Slave kann so in kurzen Intervallen mit dem Master synchronisiert werden, aber auch eine längere Zeit aussetzen um beispielsweise ein Backup aus einem, sich nicht mehr verändernden, Datenbestand zu erstellen. Der Relay-Log wird auf dem Slave serialisiert ausgeführt, was die Performance im Vergleich zur Ausführung in mehreren Threads auf dem Master schmälert. Es können alle Tabellen auf einem MySQL-Server repliziert werden oder auch nur explizit angegebene. Außerdem können Ausnahmen eingerichtet werden, um einzelne Tabellen nicht zu replizieren.

7.6.2 Skalierbarkeit

Bei der Replikation sind alle Daten auf allen beteiligten Servern vollständig vorhanden, weswegen diese auch selbstständig alle lesenden Anfragen beantworten können, ohne dabei Einfluss auf die Leistungsfähigkeit der anderen Server zu nehmen. Dadurch lässt sich die lesende Leistungsfähigkeit des Gesamtsystems durch das Hinzufügen neuer Slaves nahezu beliebig linear skalieren, unter der Annahme, dass keine schreibenden Operationen ausgeführt werden. Sobald jedoch schreibende Operationen die Datenbasis verändern, sinkt die Leistungsfähigkeit des Gesamtsystems, da der Master die geänderten Daten an die Slaves verteilen muss

52 Oracle Corporation, MySQL-Referenzhandbuch, 16.2. Replication Implementation (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-implementation.html

und die Slaves diese übernehmen müssen. Das Übernehmen der Änderungen benötigt Hardwareressourcen, die dann nicht mehr für lesende Abfragen zur Verfügung stehen und diese somit verlangsamen. Folgendes Zahlenbeispiel[53] verdeutlicht das Problem unter der Annahme, dass:

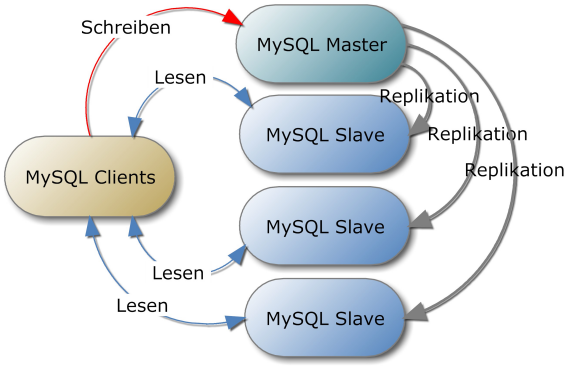

Abbildung 25: Einfache MySQL-Replikation mit 4 MySQL-Servern

Quelle: eigene Abbildung

- 80% lesende und 20% schreibende Anfragen anfallen
- Jeder Server 1.000 Abfragen pro Sekunde abarbeitet
- Schreibende Abfragen genauso aufwendig sind wie lesende

Für einen einzelnen Server würde dies also 800 lesende und 200 schreibende Abfragen ergeben. Soll nun die Leistung des Gesamtsystems durch das Hinzufügen zusätzlicher Server verdoppelt, also auf 1.600 lesende und 400 schreibende Abfragen ausgebaut werden, so müssen die 400 schreibenden Abfragen auf allen Servern ausgeführt werden. Für zwei Server müssten dann also insgesammt 2400 Abfragen verarbeitet werden, was die Leistungskapazität der beiden Server übersteigt. Wird ein dritter Server hinzugezogen so verdreifachen sich die schreibenden Anfragen, so das die Summe auf 2800 Abfragen für drei Server steigt, was innerhalb der Leistungsfähigkeit dieser liegt. Um eine Verdopplung der Leistung zu erhalten werden bei 80% lesenden und 20% schreibenden Abfragen also drei Server benötigt. Will man die Gesamtleistung noch einmal auf 3.200 lesende und 800 schreibende Abfragen verdoppeln, so werden 17 Server benötigt, um die resultierenden 16.800 Abfragen zu bearbeiten. Bei 800 Schreibenden von 1.000 Abfragen pro Server sind die Server somit also schon zu 80% mit diesen ausgelastet. Sobald 1.000 schreibende Abfragen pro Slave entstehen, ist die Replikationslösung demnach am Ende der theoretisch möglichen Skalierbarkeit angelangt, wobei aus wirtschaftlichen Aspekten dies schon viel eher der Fall sein sollte.

53 Schwartz, Baron, High Performance MySQL (2009) Seite 408

Auch wenn die Systeme selbst sehr leistungsfähig sind, kann die für die Replikation zur Verfügung stehende Bandbreite ein Problem werden. Wenn mehr Änderungen generiert werden, als an die Slaves übertragen werden können, so stauen sich diese. Wie stark eine Webapplikation mittels MySQL-Replikation skalierbar ist, hängt entscheidend von den schreibenden Operationen ab. Dem Bandbreitenproblem lässt sich entgegenwirken, indem nicht alle Slaves direkt die Änderungen vom Master, sondern von einem vertretenden Slave, lesen. Dieser vertretente Slave liest die Änderungen vom Master, führt sie aus und übernimmt sie in seinen eigenen Binärlog, so dass dieser widerum von anderen Slaves gelesen werden kann. Abbildung 26 zeigt ein Beispiel hierfür, wobei die Hierachie der Slaves beliebig aufgebaut und ergänzt werden kann. Bei diesem Beispiel sendet der Master die Daten nur einmal an einen Slave, von dem die anderen beiden Slaves die Änderungen lesen können. Es ließe sich auch eine Verkettung der Slaves von oben nach unten vornehmen, was in Anbetracht der Bandbreite günstig wäre. Dabei wird davon ausgegangen, dass ausgehender und eingehender Datentransfer unabhängig voneinander sind. Der Nachteil der Verkettung ist jedoch, dass es dabei länger dauert, bis die Änderungen auch den letzten Slave erreichen und dieser somit anfragenden Clients länger als alle anderen ggf. veraltete Informationen liefert. Mit der Verwendung mehrerer Vertreter lässt sich auch eine Baumstruktur aufbauen, was bei der Verwendung von vielen Slaves sinnvoll ist.

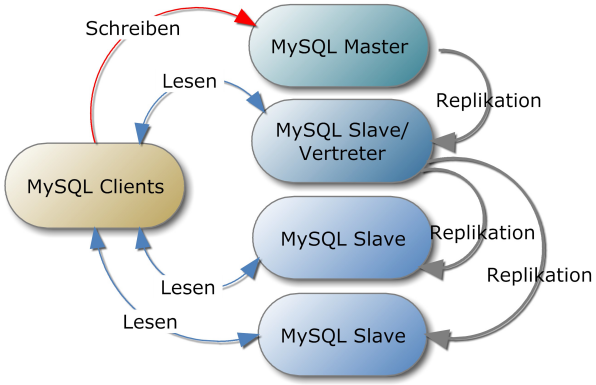

Abbildung 26: MySQL-Replikation mit vertretendem Slave

Quelle: eigene Abbildung

Eine weitere Begrenzung ist der zur Verfügung stehende Speicherplatz auf allen beteiligten MySQL-Servern. Die replizierte Datenbank kann nur so groß werden, wie der kleinste auf dem Slave zur Verfügung stehende Speicherplatz. Außerdem wird zusätzlicher Speicherplatz durch die Binär- und Relay-Logs belegt.

7.6.3 Limitierungen und Eigenheiten

Es folgt eine Übersicht der in der Praxis wichtigsten Unterschiede zwischen einem normalen MySQL-Server und einem MySQL-Replikationscluster. Diese Zusammenfassung bezieht sich auf MySQL-5.1.44.

- Bei statementbasierender Replikation liefern die SQL-Funktionen „USER()“, „CURRENT_USER()“, „UUID()“, „VERSION()“, „LOAD_FILE()“, und „RAND()“ nicht zwangsläufig auf allen Slaves dieselben Werte[54]. Bei zeilenbasierender oder gemischter Replikation funktionieren diese einwandfrei.
- NOW() verwendet einen im Binärlog gespeicherten Zeitstempel, so dass alle Slaves für NOW() bei der Replizierung dieselbe Zeit erhalten.[55] Das gilt auch für „UNIX_TIMESTAMP()“, da diese Funktion, wenn sie ohne Parameter aufgerufen wird, standardmäßig NOW() verwendet.
- Bei statementbasierender Replikation sollte die Verwendung von manuellen Tabellensperren vermieden werden, da diese den Slave ebenfalls anweisen würden, die Tabellen zu sperren.
- LIMIT ohne ORDER BY gibt auf verschiedenen Slaves potentiell andere Datensätze zurück, da die Reihenfolge, in welcher sie geliefert werden davon abhängt, wie der jeweilige Server sie intern speichert. Dies ist nur bei rein statementbasierender Replikation der Fall. Die gemische Replikation verwendet bei LIMIT ohne ORDER BY automatisch die zeilenbasierende Replikation.[56]
- Temporäre Tabellen werden repliziert. Wird der Slave aber neu gestartet, werden die temporären Tabellen verworfen und eventuell im Relay-Log befindliche Befehle, die sich darauf beziehen, schlagen nach dem Neustart fehl. Bei zeilenbasierender Replikation werden keine temporären Tabellen benötigt.
- Trigger werden auch repliziert, allerdings nur ausgeführt, wenn die Replikation

54 Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1.12. Replication and System Functions (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-features-functions.html
55 Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1.12. Replication and System Functions (2010) http://dev.mysql.com/doc/refman/5.0/en/replication-features-functions.html
56 Oracle Corporation, MySQL-Referenzhandbuch,16.4.1.13. Replication and `LIMIT` (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-features-limit.html

nicht zeilenweise abläuft.[57]

- Views werden immer repliziert.[58]
- Nicht alle in einer Session zur Verfügung stehenden Server- oder Nutzervariabeln werden korrekt statementbasierend repliziert. Mit MIXED oder zeilenbasierender Replikation kann dieses Problem umgangen werden.[59]

Wie diese Zusammenfassung erkennen lässt, bietet die zeilenbasierende bzw. gemischte Replikation teils große Vorteile gegenüber der statementbasierenden Replikation. Die obige Liste ist eine Zusammenfassung der relevantesten Besonderheiten der MySQL-Replikation. Bevor die Replikation produktiv eingesetzt wird, sollte die offizielle Dokumentation zu MySQL-Replikation konsultiert werden.[60]

7.6.4 Master-Master/Ring-Replikation - Aktiv/Aktiv

Ein besonderer Fall der Replikation ist die Master-Master-Replikation, bei der beide beteiligten Server sowohl lesbar (Slave) als auch schreibbar (Master) sind. Änderungen werden über die normale MySQL-Replikation auf den jeweils anderen MySQL-Server repliziert. MySQL-Replikation wurde ursprünglich für den Einsatz mit einem einzigen Master und mehreren Slaves konzipiert. Ein Master-Maser-Cluster kann mit Einschränkungen dennoch realisiert werden.

Der Vorteil der Master-Master-Konfiguration ist, dass die schreibenden und lesenden Abfragen der Applikationen nicht getrennt werden müssen, also jeder Server von diesen so verwendet werden kann, wie ein einzelner MySQL-Server ohne Replikation. Auch kann dies die Performance geringfügig erhöhen, da Tabellen auf beiden Servern exklusiv und unabhängig voneinander für Schreiboperationen verwendet werden können.[61] Dieser Vorteil ist gleichzeitig auch der größte Nachteil der sogenannten

57 Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1.29. Replication and Triggers (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-features-triggers.html
58 Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1.30. Replication and Views (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-features-views.html
59 Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1.32. Replication and Variables (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-features-variables.html
60 Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1. Replication Features and Issues (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-features.html
61 Oracle Corporation, MySQL-Referenzhandbuch, Replication FAQ (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-faq.html#qandaitem-17-4-4-1-5 2. Absatz

Zwei-Wege-Replikation: Dadurch, dass die Tabellensperren nur auf einem Server existieren, können, durch das Ausführen von datenverändernden Operationen in unterschiedlicher Reihenfolge, die daraus resultierenden Tabellendaten voneinander abweichen.

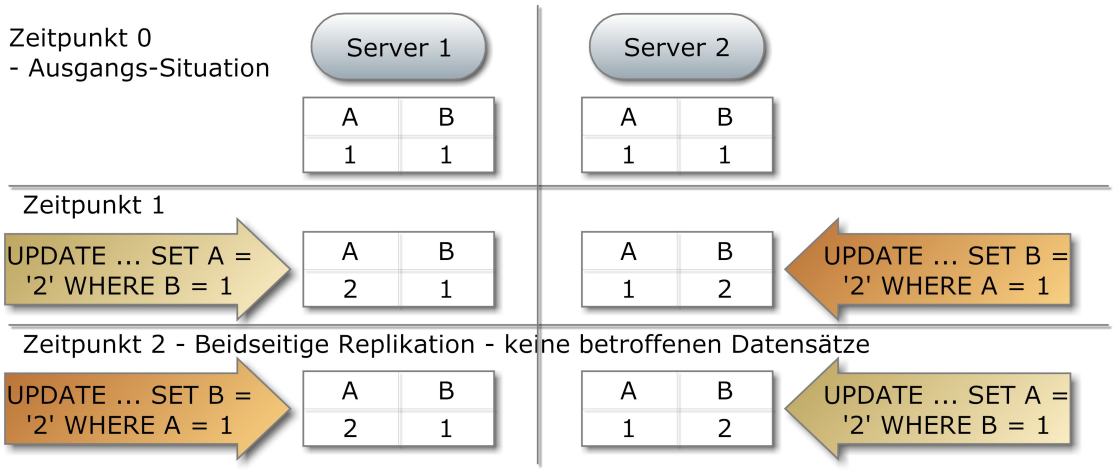

Abbildung 27: Konfliktbehaftete Updates bei Zwei-Wege-Replikation

Es entstehen jeweils voneinander abweichende Datenbanken auf beiden Servern

Quelle: eigene Abbildung nach Problembeschreibung von http://dev.mysql.com/doc/refman/5.1/en/replication-faq.html#qandaitem-17-4-4-1-5

Abbildung 27 zeigt ein drastisches Beispiel dafür. Zum Zeitpunkt 0 ist die Tabelle auf beiden Servern synchron. Dann wird zum Zeitpunkt 1 auf Server 1 ein Update ausgeführt, welches A auf 2 setzt, unter der Bedingung, dass B gleich 1 ist. Gleichzeitig wird auf Server 2 ein genau gegensätzliches Update ausgeführt, welches widerum B auf 2 setzt, wenn A gleich 1 ist. Das Replizieren dieser Updatestatements auf den jeweils anderen Server führt zu keiner Änderung, da durch das vorrangegangene Update die Bedingung nicht mehr zutrifft. Somit ist nach erfolgreicher, fehlerfreier statementbasierender Replizierung das Ergebnis inkonsistent: Die Daten auf Server 1 stimmen nicht mit Server 2 über ein. Auch die zeilenbasierende Replikation, welche die geänderten Zeilen anstelle des verändernden Befehls repliziert, würde daran nichts ändern. Das Ergebnis wäre in diesem Fall vertauscht, aber dennoch nicht auf beiden Servern gleich.

Wären serverübergreifende Tabellensperren möglich, so würde die Tabelle auf Server 1 und 2 gesperrt werden, bis das Update auf Server 1 abgeschlossen und die Daten

nach Server 2 repliziert wurden. So lange würde das Update auf Server 2 warten und anschließend mit den aktualisierten Daten, wegen der nichtzutreffenden WHERE-Bedingung, keine Daten ändern. Die Daten wären somit auf beiden Servern gleich und konsistent. Dieses Feature ist zur Zeit des Schreibens nicht verfügbar und würde auch die Leistungsfähigkeit des Gesamtsystems vermindern, da die Sperre über das Netzwerk angeordnet, bestätigt und wieder aufgehoben werden muss, was einer Verlangsamung um mehr als der doppelten Netzwerklatenz entspricht.

Ein weiteres Problem besteht beim Einfügen von neuen Datensätzen in Tabellen mit Auto-Inkrement-Feldern. Wird solch ein neuer Datensatz repliziert, so wird dabei nicht nur der Befehl selbst, sondern auch der Kontext von der Quelle zum Ziel repliziert, so dass dieser unter denselben Bedingungen ausgeführt werden kann, wie es auf dem ursprünglichen Server der Fall war. Dies schließt neben der LAST_INSERT_ID() von vorangegangenen Insert-Befehlen und der Zeit der Ausführung auch den ursprünglichen Auto-Inkrement-Wert mit ein, so dass der replizierte Befehl denselben Primärschlüssel erhält, wie auf dem ursprünglichen Server.

Die Auto-Inkrement-Variable ist für jede Tabelle auf jeden Server seperat vorhanden und wird beim Replizieren auf dem anderen Server nicht aktualisiert bzw. überschrieben.

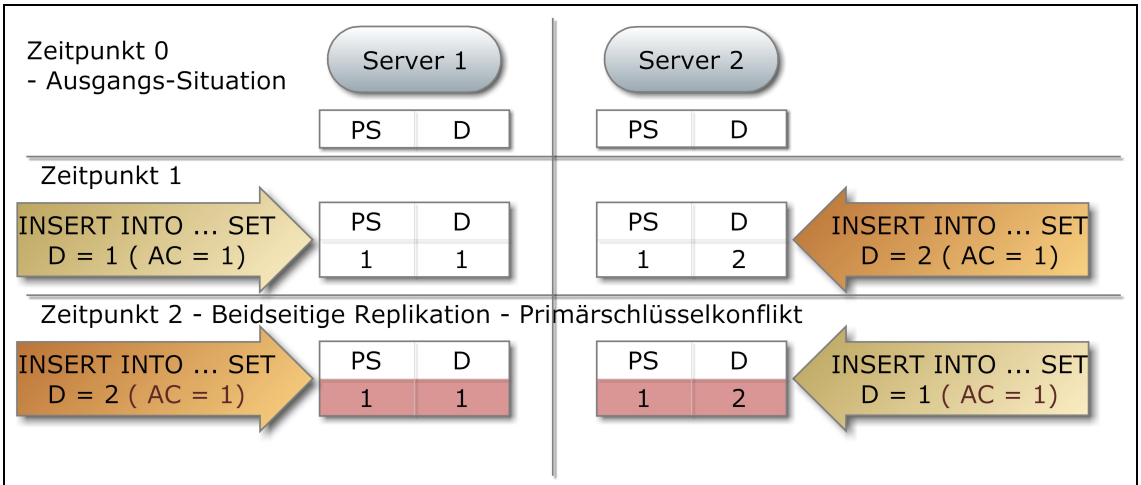

Abbildung 28: Konfliktbehaftete Inserts bei Zwei-Wege-Replikation

Die Spalte PS ist der Primärschlüssel, die Spalte D sind Nutzdaten, AC ist der Auto-Inkrement-Wert

Quelle: eigene Abbildung

Wie in Abbildung 28 zu sehen ist, entsteht durch das Replizieren des verwendeten

Auto-Inkrement-Wertes ein Primärschlüsselkonflikt auf dem Zielsystem, wodurch die Replikation fehl schlägt. Die Spalte „PS“ ist der Primärschlüssel, welcher in jeder Tabelle eindeutig sein muss. Zum Zeitpunkt 1 wird auf beiden Servern jeweils ein neuer Datensatz mit den Nutzdaten „D“ eingefügt. Die beiden neuen Datensätze erhalten, weil die Tabellen bis dahin leer waren, beide den Primärschlüssel „1“. Anschließend werden diese neuen Datensätze beidseitig zwischen den Servern repliziert. Da jedoch auf dem jeweils anderen Server schon ein Datensatz mit dem Primärschlüssel „1“ vorhanden ist, schlägt die Replikation auf den Servern fehl und wird gestoppt bis der Konflikt manuell gelöst und die Replikation wieder gestartet wurde.

Seit MySQL 5.0.2 gibt es zwei neue Servervariabeln, welche dieses Problem lösen: „auto_increment_increment“ und „auto_increment_offset“.[62] Ersteres legt die Schrittweite fest, also welche Zahl zur Generierung des nächsten Auto-Inkrement-Wertes zum vorherigen hinzuaddiert werden soll. Standardmäßig ist dies 1, für zwei Server sollte die Konfigurationsvariable auf 2 gestellt werden. Die Variable „auto_increment_offset“ bestimmt schließlich, wo mit dem Zählen begonnen werden soll. Diese kann in diesem Beispiel auf Server 1 auf 1 und auf Server 2 auf 2 gestellt werden. Dies hat zum Effekt, dass die Auto-Inkrement-Folge auf Server 1 {1,3,5,7} und auf Server 2 {2,4,6,8} lautet, wodurch keine Konflikte mehr entstehen.

Es ist möglich, mehr als zwei Knoten als Master und gleichzeitig als Slave zu konfigurieren. Dabei entsteht ein Ring (s. Abb. 29), in dem jeder Master von einem Slave die Daten repliziert. Dabei müssen alle Server so konfiguriert werden, dass sie neben den eigenen, neuen schreibenden Operationen auch die der vorherigen Server weitergeben, so das jedes Event den gesamten Ring durchläuft. Sobald ein Event wieder beim ursprünglichen Sender angekommen ist, wird es anhand der Server-ID verworfen, so dass ein endloses Replizieren im Ring

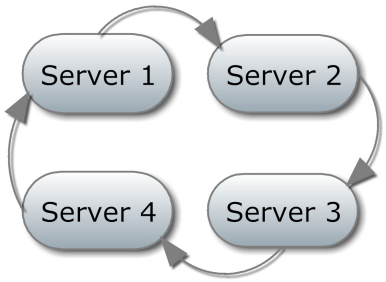

Abbildung 29: Ring-Replikation mit vier Knoten

Jeder Knoten fungiert sowohl als Master als auch als Slave

Quelle: eigene Abbildung

62 Oracle Corporation, MySQL-Referenzhandbuch, Replication Options Master (2010) http://dev.mysql.com/doc/refman/5.0/en/replication-options-master.html#sysvar_auto_increment_increment

vermieden wird. Fällt jedoch ein Server aus, der gerade Events gesendet hat, so ist dieser nicht mehr verfügbar, um die eigenen Events zu erkennen und zu entfernen, weswegen sie endlos im Ring repliziert und ausgeführt werden. Eine saubere Möglichkeit, diese zirkulierenden Events zu entfernen, gibt es zur Zeit nicht, allerdings ist die Option „IGNORE_SERVER_IDS“ des „CHANGE MASTER TO“-Kommandos in der Entwicklerversion 5.5 dokumentiert[63], welche dieses Problem lösen soll.

7.6.5 Failover

Der MySQL-Server mit Replikation unterstützt kein automatisches Failover, bietet also weder eine Fehlererkennung noch eine automatische Rekonfiguration des Clusters an. Ein einfaches Failover lässt sich mit MySQL-Bordmitteln in Form von Prozeduren und zeitlich gesteuerten MySQL-Events programmieren[64], was jedoch durch die, im Vergleich zu einer Systemprogrammiersprache eingeschränkte Funktionalität von MySQL unflexibel und problematisch sein kann. Eine fortgeschrittene Überwachung und eine Fehlererkennung, das eigentliche Failover und auch eine eventuelle Selbstheilung, müssen also über externe Programme bzw. Skripte realisiert werden.

Da MySQL-Replikation eine sehr beliebte Methode ist, um eine MySQL-Datenbank ausfallsicher und skalierbar zu machen, existieren für diesen Zweck bereits eine handvoll Programme, die das Failover automatisieren. Der bekannteste Vertreter dürfte der „Multi-Master Replication Manager for MySQL“[65] oder kurz MMM sein. Außerdem sollte an dieser Stelle der „Maatkit“[66] genannt werden. Dabei handelt es sich um eine Skriptsammlung, welche Funktionalität bereitstellt, die dem MySQL-Server noch fehlt und bei Problemen sehr hilfreich sein kann. Unter anderem umfasst Maatkit Skripte zum Überprüfen der Intigrität von MySQL-Tabellen zwischen Master und Slave als auch ein Skript, welches zwei Tabellen auf effiziente Art synchronisieren kann. Diese Funktionalität ist sehr praktisch, um Fehler bei der Replikation zu erkennen und zu beheben. Ob und welche Software das Failover steuert, kommt auf die Gegebenheiten und das Setup an in dem die Server betrieben werden.

63 Oracle Corporation, MySQL-Referenzhandbuch, 12.5.2.1. `CHANGE MASTER TO` Syntax (2010) http://dev.mysql.com/doc/refman/5.5/en/change-master-to.html
64 O'Reilly Media, Inc., Automatic Fail-Over(2010), http://onlamp.com/pub/a/onlamp/2006/04/20/advanced-mysql-replication.html?page=4
65 http://mysql-mmm.org/start
66 http://www.maatkit.org/

7.6.6 Lesen und Schreiben trennen

Ein typisches Problem bei allen Arten der MySQL-Replikation, abgesehen von der Aktiv/Aktiv-Master/Master-Replikation, ist, dass schreibende Abfragen nur an den Master gesendet werden dürfen, während lesende vorzugsweise auf den Slaves ausgeführt werden sollen. In der englischen Literatur wird für dieses Problem der Begriff „*Read/Write-Splitting*“ bzw. „*RW-Splitting*“ verwendet, welche im Weiteren verwendet werden. Es existieren zwei grundlegende Lösungsansätze: zum Einen applikationsbasierend, zum Anderen über eine Art Proxy.

Bei der applikationsbasiernden Trennung werden die lesenden und schreibenden SQL-Abfragen von der (Web-) Applikation an verschiedene Server geschickt, was die sauberste Lösung darstellt, sofern die Applikation dies unterstützt. Implementiert wird dies oft, indem innerhalb des Applikationscodes jeweils eine Funktion für lesende und eine Funktion für schreibende Datenbankoperationen verwendet wird, so dass der Programmierer der Applikation entscheidet, welche Befehle auf dem Master und welche auf dem Slave ausgeführt werden.

Sollte die Applikation dies nicht unterstützen und sich diese Funktionalität auch nicht mit vertretbaren Aufwand nachrüsten lassen, so bleibt nur die Möglichkeit, die Befehle auf dem Weg von der Applikation zum SQL-Server umzuleiten. Die Unterscheidung erfolgt dabei anhand der Befehle. Die einfachste Variante ist, alle Befehle, die mit „SELECT“ beginnen an die Slaves zu leiten und alle Anderen an den Master.

Eine applikationsunabhängie Implementierung ist das „*Read/Write-Splitting*“-Plugin für MySQL-Proxy. MySQL-Proxy ist eine recht simple Applikation, welche zwischen MySQL-Server und die Clients geschalten wird. Ähnlich wie bei einem klassischen HTTP-Proxy versteht MySQL-Proxy das MySQL-Kommunikations-Protokoll und agiert für die Clients als Server und für den eigentlichen MySQL-Server als Client. Die Software kann neben dem *RW-Splitting* auch zur Analyse, Filterung und Veränderung von SQL-Abfragen als auch für Failover und Lastverteilung verwendet werden.

Die Implementierung des *RW-Splitting*-Plugins für MySQL-Proxy ist z.Z. nur ein Machbarkeitsnachweis und aufgrund von Einschränkungen noch keine für jeden Anwendungszweck geeignete Lösung.[67]

67 Unbekannt, MySQL Proxy RW Splitting (2010),

7.6.7 Aufbau des Testsetups

Für ein Setup mit zwei Servern kommen prinzipiell zwei Ansätze in Frage: einerseits die einfache Master-Slave-Replikation, andererseits die Master-Master-Replikation. Auch wenn die Master-Master-Replikation es erlaubt, dass beide Server aktive, schreibbare Komponenten des Clusters sind, so müssen nicht zwangsläufig auf beiden Servern schreibende Befehle ausgeführt werden. Wird nur ein Server, wie bei der Master-Slave-Replikation, für schreibende Anfragen verwendet, so können die Replikationskonflikte, wie in Kapitel 7.6.4 „Master-Master/Ring-Replikation - Aktiv/Aktiv“ beschrieben, vermieden werden. Das Wechseln des aktiven Servers bei einer geplanten Serverwartung oder einem Ausfall gestaltet sich in diesem Fall deutlich einfacher als bei der Master-Slave-Replikation: Statt die Replikation auf dem bisherigen Slave zu stoppen und sie auf dem bisherigen Master wieder zu initialisieren und anschließend alle Anfragen umzuleiten, müssen so nur die Anfragen umgeleitet werden, da die beidseitige Replikation bereits besteht. Es kann auch bei der Master-Master-Replikation ein Server als „Nur-Lesend“ deklariert werden, um versehentliche Schreiboperationen zu blockieren. Aufgrund dieser Vorteile wird im folgenden die Master-Master-Replikation getestet.

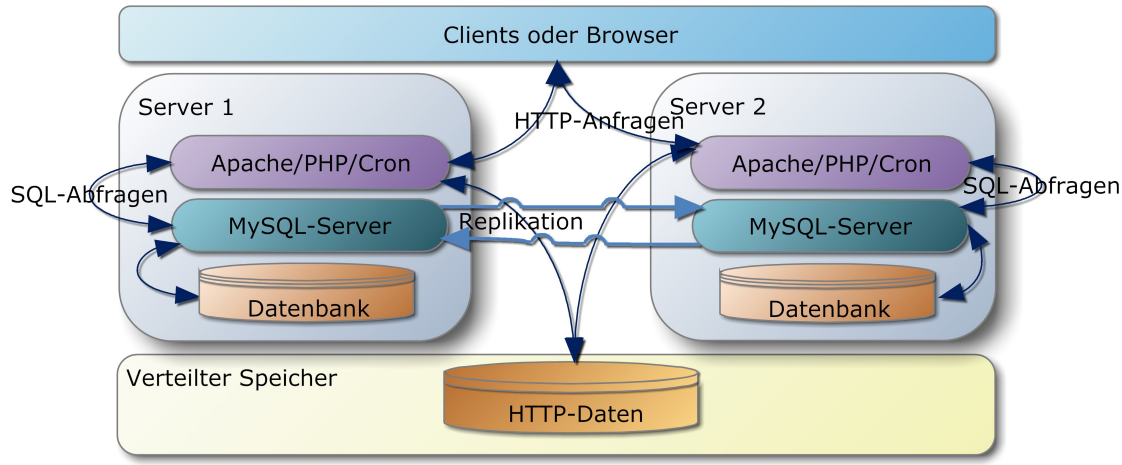

Abbildung 30: Aufbau eines Master-Master MySQL-Replikations-Clusters mit zwei Servern
Die helleren blauen Pfeile symbolisieren die beidseitige Replikation.
Quelle: eigene Abbildung

http://forge.mysql.com/wiki/MySQL_Proxy_RW_Splitting#Limitations

Mit der Master-Master-Replikation lässt sich auch eine einfache projektbasierende Lastverteilung realisieren: Je nach Ressourcenbedarf können die einzelnen Kundenprojekte auf die vorhandenen Server verteilt werden, so dass ein Teil der Projekte auf dem einen und der Rest auf dem anderen Server schreibbar ist. Dies hat den Vorteil, das keine Trennung von lesenden und schreibenden Anfragen nötig und dennoch ein gewisses Maß an Lastverteilung möglich ist. Die Hochverfügbarkeit der Projekte ist ebenfalls gesichert, da die Replikate jederzeit genutzt werden können. Die Lastverteilung und auch die Hochverfügbarkeit könnten so mit einem einfachen HTTP-Lastverteiler realisiert werden.

7.6.8 Installation und Konfiguration

Da die Replikationsfähigkeit bei der Standarddistribution von MySQL bereits vorhanden ist, verläuft die Einrichtung eines Replikationsclusters entsprechend einfach. Die Installation, Konfiguration und das Einspielen der Daten kann exakt so erfolgen wie in Kapitel 7.3 „Standard MySQL-Server“ beschrieben.

7.6.8.1 Einrichten der Master-Master-Replikation

Nachdem auf beiden Servern der MySQL-Server installiert und die Datenbank eingespielt wurde, muss nun die Replikation konfiguriert werden.

Die MySQL-Hauptkonfigurationsdatei /etc/my.cnf wird entsprechend Abbildung 31 angepasst. In einem Replikationssetup benötigt jeder Server eine eindeutige Identifikationsnummer im Bereich von 1 bis $2^{32}-1$.[68] Diese Nummer wird mit dem Konfigurationsparameter „server_id“ festgelegt und muss folglich auf Server 2 auf einen anderen Wert als 1, also z.B. 2 festgelegt werden. Die darauf folgenden Parameter „log_bin“, „log_bin_index“,

```
[mysqld]
query_cache_type = 0

server_id  = 1
log_bin  = /var/log/mysql/bin.log
log_bin_index  = /var/log/mysql/bin.log.idx
relay_log = /var/log/mysql/relay-bin
relay_log_index = /var/log/mysql/relay-bin.idx
max_binlog_size = 100M
expire_logs_days = 7
```

Abbildung 31: /etc/my.cnf für Replikation

Quelle: eigene Konfigurationsdatei

68 Oracle Corporation, MySQL-Referenzhandbuch, 16.1.3. Replication and Binary Logging Options and Variables (2010), http://dev.mysql.com/doc/refman/5.1/en/replication-options.html#option_mysqld_server-id

„relay_log“ und „relay_log_index“ aktivieren den Binär- bzw. den Relay-Log und legen die entsprechenden Dateipfade fest. Die Index-Dateien enthalten die Namen aller verwendeten Binärlog-Dateien. Mit „max_binlog_size“ wird die Maximalgröße der Binärlog-Dateien von standardmäßig einem Gigabyte auf 100 Megabyte festgelegt. Wird diese Grenze überschritten, legt der MySQL-Server eine neue Binärlogdatei an und schreibt von da an in dieser weiter. Damit der Speicherverbrauch der Binärlogs nicht unendlich wächst, legt „expire_logs_days“ fest, nach wievielen Tagen die Binärlogdateien gelöscht werden können.

Nach dem Starten beider MySQL-Server wird die Konfiguration an den laufenden Systemen fortgesetzt. Server 1 besitzt die IP 192.168.0.210 und Server 2 192.168.0.220 wie schon beim MySQL-Cluster-Setup. Zuerst muss auf beiden Servern ein Nutzer mit Replikationsrechten angelegt werden. Dies geschieht mit dem MySQL-Befehl:

- GRANT REPLICATION SLAVE ON *.* TO 'replication'@'192.168.100.%' IDENTIFIED BY 'password';

Anschließend, da die Datenbanken auf beiden Servern neu eingespielt wurden und genau gleich sind, kann die Replikation konfiguriert werden. Hierfür ist die aktuelle Logposition und Datei des jeweiligen Masters wichtig, welche sich durch folgenden Befehl in Erfahrung bringen lässt:

- SHOW MASTER STATUS;

Dieser muss auf beiden Servern ausgeführt werden, da die Logposition und Logdatei eventuell unterschiedlich sein könnten. Anschließend wird die Replikation auf beiden Servern konfiguriert:

- Server 1: CHANGE MASTER TO master_host='192.168.0.220', master_port=3306, master_log_file='mysql-bin.000002', master_log_pos=106, master_user='replication', master_password='password';
- Server 2: CHANGE MASTER TO master_host='192.168.0.210', master_port=3306, master_log_file='mysql-bin.000002', master_log_pos=106, master_user='replication', master_password='password';

Mit der Ausführung des MySQL-Befehls „SLAVE START“ auf beiden Servern wird die

wechselseitige Replikation gestartet. Von da an werden alle Änderungen zwischen beiden Servern ausgetauscht, so dass die Tests nun durchgeführt werden können.

7.6.9 Testergebnisse

	Lauf 1 in s	***Lauf 2 in s***	***Lauf 3 in s***	***Lauf 4 in s***	***Lauf 5 in s***	***Durchsch. Lauf 1-5***	***Standard-Mysql in s***	***Änderung in %***
Test 1	37,64	37,73	37,22	37,38	37,81	** Fehlerhafter Ausdruck **	37,38	** Fehlerhafter Ausdruck **
Test 2	11,67	11,55	11,81	12,05	11,06	** Fehlerhafter Ausdruck **	11,97	** Fehlerhafter Ausdruck **
Test 3	64,54	64,39	65,35	64,14	63,91	** Fehlerhafter Ausdruck **	143,46	** Fehlerhafter Ausdruck **
Test 4	33,68	33,12	32,32	31,01	33,88	** Fehlerhafter Ausdruck **	31,26	** Fehlerhafter Ausdruck **
Test 5	116,35	116,5	115,35	118,15	115,78	** Fehlerhafter Ausdruck **	123,1	** Fehlerhafter Ausdruck **
Test 6	34,59	34,31	34,18	34,39	34,94	** Fehlerhafter Ausdruck **	34,25	** Fehlerhafter Ausdruck **
Summe						** Fehlerhafter Ausdruck **	** Fehlerhafter Ausdruck **	** Fehlerhafter Ausdruck **

Tabelle 7: Testergebnisse der MySQL-Replikation

Quelle: eigene Berechnungen und Messungen

Test 1 und 2 fallen, wie zu erwarten war, ohne einen signifikanten

Geschwindigkeitsunterschied aus, da es sich bei diesen beiden Tests um lesende Abfragen auf einem einzelnen Server handelt. Dafür entsteht bei Test 3 eine Leistungsverbesserung von 55%, was den Erwartungswert von 50% leicht übertrifft. Test 4, das Schreiben auf einem einzelnen Server, ist nur minimal langsamer als beim Standard-MySQL-Server, was durch den zusätzlichen Aufwand beim Senden der Änderungen an den anderen Server zu erklären wäre. Test 5, die Vierfache Ausführung von Test 4 ist mit ca. 5% nicht wesentlich schneller als auf einem einzelnen Server. Bei Test 6 verhält es sich ähnlich wie bei Test 1 und 2, da hier wieder nur lesende Abfragen an einen einzelnen Server entstehen. Die Messergebnisse bestätigen die in der theoretischen Betrachtung erarbeiteten Thesen zur Skalierbarkeit von lesenden und schreibenden Abfragen.

7.6.10 Einschätzung

Im Vergleich zum Standard-MySQL-Server ergibt sich ein Geschwindigkeitsvorteil bei lesenden, aber kein Nachteil bei schreibenden Abfragen. Dadurch ist diese Lösung gut geeignet, um eine hohe Verfügbarkeit und dennoch eine gewisse Skalierbarkeit zu erreichen. Dadurch, dass diese Lösung aufgrund von schreibenden Abfragen nicht unendlich skalierbar ist, muss ab einer gewissen Projektgröße eine andere Lösung gefunden werden, beispielsweise durch Optimierung der eingesetzten Web-applikationen auf MySQL-Cluster. Für kleinere Projekte mit Standard-Webapplikationen ist die MySQL-Replikation jedoch eine gute Möglichkeit, diese etwa auf die zwei- bis zehnfache Leistungsfähigkeit zu skalieren.

8 Fazit und Gesamtkonzept

In Kapitel 4 „Theoretische Konzepte für Hochverfügbarkeits-LAMP“ wurde festgestellt, dass Lösungen, welche auf virtuellen Maschinen basieren, den Anforderungen aus Kapitel 3.3 , speziell was die Skalierbarkeit angeht, nicht gerecht werden, weswegen nur eine applikationsbasierende Lösung in Frage kommt. Dies bedeutet, dass sowohl der Webserver, die Datenbank und der E-Mail-Server redundant und hochverfügbar gestaltet werden müssen. Die Hochverfügbarkeit des E-

Mailservers lässt sich unter Verwendung von mehreren MX-Einträgen in der Domain und einem gemeinsamen Speicher, wie in Kapitel 6.2 „Mailserver mit gemeinsamen Speicher“ beschrieben, erreichen. Um die MySQL-Datenbank auf zwei Server zu verteilen und somit Hochverfügbarkeit und Skalierbarkeit zu erreichen, wurde in Kapitel 7 die MySQL-Replikation als akzeptable Lösung für das Unternehmen ermittelt.

8.1 Gesamtkonzept für das Unternehmen

Das Ergebnis der vorrangegangenen Untersuchungen ist nun ein Gesamtkonzept, welches die gestellten Anforderungen weitestgehend erfüllt. Einzige Ausnahme ist, dass die aktuell gemietete Serverhardware nicht weiter verwendet werden kann, da Failover-IP-Adressen benötigt werden, welche im aktuellen Rechenzentrum nicht zur Verfügung stehen. Ein Umzug in ein anderes Rechenzentrum, welches dieses Feature bereitstellt, ist nach Absprache mit dem Unternehmen kein Problem.

Zur Realisierung der Lösung werden zwei Server benötigt, welche sich im selben Rechenzentrum befinden müssen, so dass zwei Failover-IP-Adressen zwischen diesen umgeschaltet werden können. Die zentrale Komponente der Lösung ist der MySQL-Datenbankserver, ohne welchen die Realisierung nicht möglich wäre. Als Replikationstopologie kommt die Master-Master-Replikation zum Einsatz, wobei diese auf beiden Servern prinzipiell schreibbar ist. Um Replikationskonflikte zu vermeiden, werden die Kundenprojekte in zwei Gruppen aufgeteilt. Die Aufteilung erfolgt anhand des zu erwarteten Ressourcenbedarfs, so dass beide Gruppen in der Summe möglichst gleich viele Ressourcen benötigen. Die Domains der Kundenprojekte werden so konfiguriert, dass alle aus Gruppe 1 auf die Failover-IP von Server 1 und alle aus Gruppe 2 auf die von Server 2 referenzieren. Je nach Gruppenbildung kann so die Last gleichmäßig auf beide Server verteilt werden, wobei jede Domain exklusiv auf einem Server den Besuchern zugänglich gemacht wird. Dies verhindert, dass beispeilsweise ein Online-Shop auf beiden Servern gleichzeitig verwendet wird und dadurch die in Kapitel 7.6.4 „Master-Master/Ring-Replikation - Aktiv/Aktiv“ beschriebenen Replikationsprobleme entstehen können. Andererseits bedeutet dies, dass insgesamt die Kapazitäten von beiden Servern für die Kundenprojekte zur Verfügung stehen. Der Aufbau der Lösung wird in Abbildung 32 dargestellt.

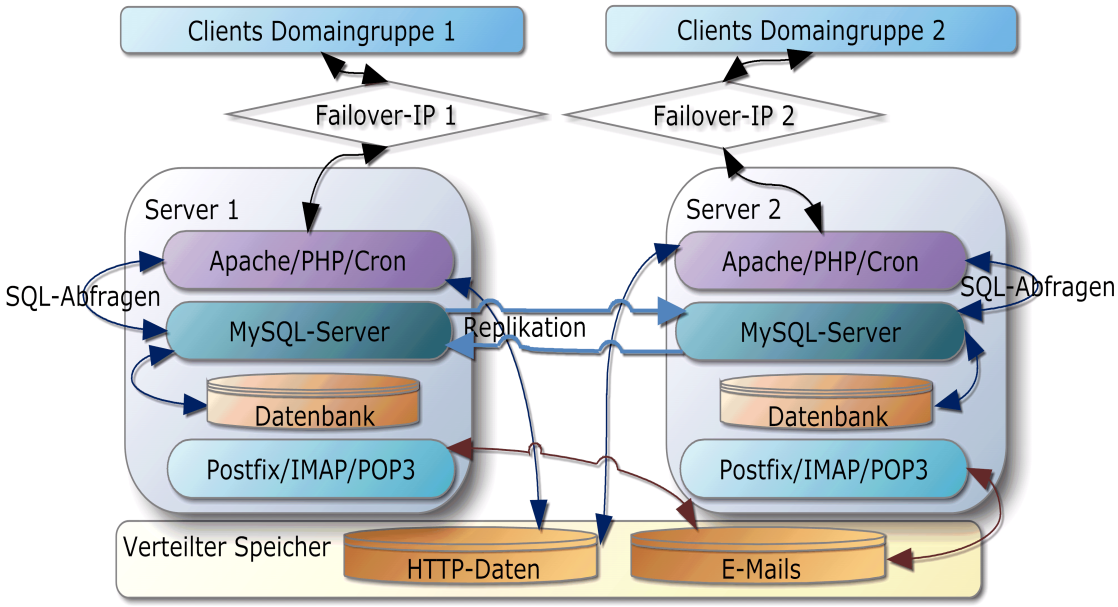

Abbildung 32: Hochverfügbarkeitssetup mit zwei Failover-IPs und Domain-Gruppen

Quelle: eigene Abbildung

Beide Webserver verwenden die auf einem gemeinsamen Speicher bereitgestellten HTTP-Daten, wobei auch die PHP-Session-Informationen auf diesem oder in der Datenbank gespeichert werden müssen, so dass die Session eines Besuchers auf beiden Servern gleichermaßen zur Verfügung steht. Auch beide Postfix-E-Mail-Server und die dazugehörigen IMAP- und POP3-Server verwenden den gemeinsamen Speicher zum Speichern und Abrufen der E-Mails. Dadurch stehen die E-Mails den Nutzern auf beiden Servern zur Verfügung, auch wenn einer ausfallen sollte. Auch können beide E-Mail-Server Nachrichten unabhängig voneinander entgegennehmen, so dass dadurch auch unter Verwendung von mehreren MX-Adresseinträgen in der Domain eine Lastverteilung erreicht werden kann.

8.1.1 Failover

Im Falle eines Serverausfalls wird die Failover-IP des ausgefallenen Servers auf den verbliebenen umgeschwenkt, so dass ab nun alle Anfragen von diesem verarbeitet werden. Das Ändern des Ziels einer Failover-IP erfolgt nahezu sofort und besitzt die Eigenschaft, dass zu keinem Zeitpunkt Besucher sowohl an Server 1 und an Server 2 geleitet werden. Damit werden sowohl die Webapplikationen als auch der E-Mail-

Server hochverfügbar gestaltet. Bestehende TCP-Verbindungen werden dabei zwar unterbrochen, können aber sofort wieder aufgebaut werden. Die Erkennung eines Serverausfalles und die Umstellung der Failover-IP muss automatisch erfolgen. Möglicherweise bietet der Serveranbieter dafür schon eine fertige Lösung in Form eines Beobachtungsdienstes an, welcher die Verfügbarkeit prüft und automatisch die Failover-IP umschaltet. Ist dies nicht gegeben, so muss dies unter Verwendung der Programmierschnittstelle für die Failover-IP programmiert werden. Es bietet sich an, das Beobachten der Serververfügbarkeit von einem Rechner außerhalb des Rechenzentrums durchzuführen, da so das *Split-Brain*-Problem vermieden werden kann.[69]

Ein generelles Problem bei einem Serverausfall ist, dass möglicherweise nicht alle Daten auf dem ausgefallenen Server gespeichert wurden und verloren gehen. Auch kann es passieren, dass die kurz vor dem Ausfall ausgeführten Änderungen an der Datenbank noch nicht auf den anderen Server übertragen wurden. Die Wahrscheinlichkeit, dass dies relevante Daten betrifft und dadurch ein wirtschaftlicher Schaden entsteht, wird aufgrund der Beschaffenheit der Kundenprojekte als gering eingeschätzt. Da nach Ausfall eines Servers alle Zugriffe nur noch auf den verbliebenen Server erfolgen, kann es zu Leistungsengpässen kommen. Es ist möglich, dass ein einzelner Server mit den Anfragen überlastet ist und somit langsamer als üblich antwortet. Deswegen sollte bei der Planung beachtet werden, dass im Notfall ein einzelner Server alle Anfragen in noch vertretbarer Zeit beantworten kann. Auch muss klar sein, dass ein Failover-Setup zwar effektiv einen vollständigen Serverausfall überbrückt, ein regelmäßiges Backup jedoch nicht ersetzt. Durch Schäden an der Hardware, Bedienfehler oder Ähnliches können Daten beschädigt werden oder verloren gehen. Diese Schäden sind auf dem gemeinsamen Speicher oder nach erfolgter Replikation auf beiden Servern präsent.

8.1.2 Wiedereingliederung des ausgefallenen Servers

Sobald der ausgefallene Server wieder funktionsfähig ist, sollte dieser wieder in das Hochverfügbarkeitssetup eingegliedert werden. Dies wird manuell von einem Administrator erledigt, da dieser bei möglichen Problemen schnell und flexibel

69 Siehe dazu Kapitel 7.5.1.4 „Die Rolle des Arbitrators“

reagieren kann. Er muss sicherstellen, dass, nach Synchronisation mit dem Relay-Log des verbliebenen Servers, beide Datenbanken konsistent sind. Ist dies nicht der Fall, so müssen vor der Wiederaufnahme der Replikation die Datenbanken synchronisiert werden. Nach dem Start der Replikation und nach dem Wiedereinbinden des gemeinsamen Speichers wird die Failover-IP wieder auf diesen Server gestellt, so dass ab da wieder Anfragen eintreffen. Prinzipiell ist es möglich, diese Schritte zu automatisieren. Da es sich aber um eine neue Lösung handelt, wird dies vorerst ein manueller Prozess bleiben, um Fehler und Probleme zu vermeiden.

8.2 Fazit

Die hier vorgestellte Lösung wurde vom Unternehmen akzeptiert. Die Umsetzung ist für das 2. Quartal 2011 im Zuge eines bereits angedachten Wechsel des Serveranbieters geplant. Die gefundene Lösung bietet einen guten Kompromiss zwischen Kosten und sich bietenden Vorteilen. Neben der Miete für die Server entsteht ein überschaubarer administrativer Aufwand, so dass diese Hochverfügbarkeitslösung zu günstigen Konditionen den Kunden des Unternehmens angeboten werden kann. Vor dem produktiven Einsatz der Lösung ist eine zweiwöchige Testphase geplant, bei welcher diese intensiv getestet wird. Sollten mehr Serverkapazitäten benötigt werden, ist es denkbar, die hier vorgestellte Lösung auch mehrmals umzusetzen, wobei immer zwei Server ein Paar bilden.

a) Anhang

1 Test 3: Komplexes SQL-Query

```
SELECT * FROM
(
	SELECT
		count( * ) AS "in x Gruppen",
		p.products_model,
		p.products_ean,
		pd.products_name,
		p.products_price,
		group_concat( c.categories_path ) AS "Gruppenpfade",
		c.categories_id, c.categories_path,
		p.products_availability_id

	FROM
		products p,
		products_description pd,
		products_to_categories ptc,
		categories c,
		categories_description cd
	WHERE
		p.products_id = ptc.products_id
		AND p.products_id = pd.products_id
		AND ptc.categories_id = c.categories_id
		AND c.categories_id = cd.categories_id
		AND pd.language_id =2
		AND cd.language_id =2
		AND
		(
			p.products_status =1
			OR p.products_add_to_amazon =1
		)
		AND p.products_availability_id =0
		AND p.products_ean IS NOT NULL
		AND CHAR_LENGTH( p.products_ean ) >7
		AND pd.products_name NOT LIKE '%Nachweispflicht beachten%'
		AND NOT
		(
			p.products_model LIKE '%DC%'
			AND pd.products_name LIKE '%EDU%'
		)
		AND p.products_price > 0
	GROUP BY
		p.products_model
	ORDER BY
		p.products_availability_id,
		p.products_price
) AS tmp
GROUP BY tmp.products_ean LIMIT 1
```

b) Literaturverzeichnis

Printmedien

Schwartzkopff, Michael, Clusterbau mit Linux-HA Version 2, 1. Auflage (2008) ISBN 978-3-89721-779-9

Baron Schwartz, Peter Zaisev, Vadim Tkachenko, Jemery D. Zawodny, Arijen Lentz & Derek J. Balling, High Performance MySQL 2. Auflage (2009), Kösel, ISBN 978-3-89721-889-5

Internetquellen

Kunze, Michael, c't 12/98, Seite 230 - LAMP: Freeware Web Publishing System with Database Support (1998), http://web.archive.org/web/20071212203835/http://www.heise.de/ct/english/98/12/230/

Netcraft Ltd, April 2010 Web Server Survey (2010), http://news.netcraft.com/archives/2010/04/15/april_2010_web_server_survey.html

Oracle Corporation, Market Share (2010), http://www.mysql.com/why-mysql/marketshare/

TIOBE Software BV, TIOBE Programming Community Index for July 2010 (2010), http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

HowtoForge, How To Set Up A Ubuntu/Debian LAMP Server, http://www.howtoforge.com/ubuntu_debian_lamp_server

Dupke, Kai, Der Weg zu Hochverfügbarkeitslösungen (2001), http://research.intelego.net/heise/ix/2001/04/089/art.htm

Bundesamt für Sicherheit in der Informationstechnik, HV-Kompendium V1.2 - Datenbanken (2009), Seite 32, https://www.bsi.bund.de/cae/servlet/contentblob/483628/publicationFile/30947/9_Datenbanken_pdf.pdf

The Open Group Base, Specifications Issue 7, Crontab (Datei) (2008), http://www.opengroup.org/onlinepubs/9699919799/utilities/crontab.html

Oracle Corporation, MySQL Referenzhandbuch, 2.4.2. Operating Systems Supported by MySQL Community Server (2010), http://dev.mysql.com/doc/refman/5.0/en/which-os.html

Heise Online, Sun kauft MySQL AB für eine Milliarde US-Dollar (2008) http://www.heise.de/newsticker/meldung/Sun-kauft-MySQL-AB-fuer-eine-Milliarde-US-Dollar-Update-179176.html

Golem.de, Oracle kauft Sun (2009), http://www.golem.de/0904/66578.html

Oracle Corporation, Über MySQL (2010), http://www.mysql.de/about/

Oracle Corporation, MySQL-Customers (2010), http://www.mysql.com/customers/ und unterseiten.

Oracle Corporation, MySQL-Referenzhandbuch, 1.8.5. MySQL Differences from Standard SQL (2010)

International Organization for Standardization, ISO/IEC 9075-1:2008 (2010), http://www.iso.org/iso/catalogue_detail.htm?csnumber=45498

Oracle Corporation, MySQL-Referenzhandbuch, 11.2. Numerische Datentypen (2010), http://dev.mysql.com/doc/refman/5.1/de/numeric-types.html

Oracle Corporation, MySQL-Referenzhandbuch, 7.7.2. Table Locking Issues (2010) http://dev.mysql.com/doc/refman/5.0/en/table-locking.html

Reisner, Philipp und Ellenberg, Lars, DRBD v8 - Replicated Storage with Shared Disk Semantics (2007), http://www.drbd.org/fileadmin/drbd/publications/drbd8_wpnr.pdf

Dynamic Network Services Inc, DNS Caching (2010), http://www.dyndns.com/support/kb/dns_caching.html

JH Software ApS, DNS Caching and Simple Failover (2006) Seite 2 http://www.simplefailover.com/outbox/dns-caching.pdf

Depper, Ulrich, nscd and DNS TTL (2007) http://udrepper.livejournal.com/16362.html

Microsoft, How Internet Explorer uses the cache for DNS host entries (2009) http://support.microsoft.com/default.aspx?scid=KB;en-us;263558

JH Software ApS, DNS Caching and Simple Failover (2006) Seite 3 http://www.simplefailover.com/outbox/dns-caching.pdf

JH Software ApS, DNS Caching and Simple Failover (2006) Seite 3 http://www.simplefailover.com/outbox/dns-caching.pdf

Refsnes Data, Browser Statistics (2010), http://www.w3schools.com/browsers/browsers_stats.asp

Bernstein, Daniel, Using maildir format (1995), http://cr.yp.to/proto/maildir.html

Oracle Corporation, MySQL-Referenzhandbuch, 14.1. Using MySQL with DRBD (2010), http://dev.mysql.com/doc/refman/5.1/en/ha-drbd.html

Oracle Corporation, MySQL-Referenzhandbuch, Chapter 14. High Availability and Scalability, http://dev.mysql.com/doc/refman/5.1/en/ha-overview.html

Bellard, Fabrice, About QEMU (2010), http://wiki.qemu.org/Main_Page

Oracle Corporation, MySQL-Referenzhandbuch,7.3.1.8. Nested-Loop Join Algorithms (2010) http://dev.mysql.com/doc/refman/5.1/en/nested-loop-joins.html

Helm, Bernd, Ndb crashing at startup (2010), http://bugs.mysql.com/bug.php?id=54310

Oracle Corporation, MySQL-Referenzhandbuch, FAQ (2010), http://dev.mysql.com/doc/refman/5.1/en/replication-faq.html#qandaitem-17-4-4-1-13 http://dev.mysql.com/doc/refman/5.1/en/replication-formats.html

Oracle Corporation, MySQL-Referenzhandbuch, 16.2. Replication Implementation (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-implementation.htm

Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1.12. Replication and System Functions (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-features-functions.html

Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1.12. Replication and System Functions (2010) http://dev.mysql.com/doc/refman/5.0/en/replication-features-functions.html

Oracle Corporation, MySQL-Referenzhandbuch,16.4.1.13. Replication and LIMIT (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-features-limit.html

Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1.29. Replication and Triggers (2010), http://dev.mysql.com/doc/refman/5.1/en/replication-features-triggers.html

Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1.30. Replication and Views (2010), http://dev.mysql.com/doc/refman/5.1/en/replication-features-views.html

Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1.32. Replication and Variables (2010). http://dev.mysql.com/doc/refman/5.1/en/replication-features-variables.html

Oracle Corporation, MySQL-Referenzhandbuch, 16.4.1. Replication Features and Issues (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-features.html

Oracle Corporation, MySQL-Referenzhandbuch, Replication FAQ (2010) http://dev.mysql.com/doc/refman/5.1/en/replication-faq.html#qandaitem-17-4-4-1-5 2. Absatz

Oracle Corporation, MySQL-Referenzhandbuch, Replication Options Master (2010) http://dev.mysql.com/doc/refman/5.0/en/replication-options-master.html#sysvar_auto_increment_increment

Oracle Corporation, MySQL-Referenzhandbuch, 12.5.2.1. CHANGE MASTER TO Syntax (2010) http://dev.mysql.com/doc/refman/5.5/en/change-master-to.html

O'Reilly Media, Inc., Automatic Fail-Over(2010), http://onlamp.com/pub/a/onlamp/2006/04/20/advanced-mysql-replication.html?page=4

Unbekannt, MySQL Proxy RW Splitting (2010), http://forge.mysql.com/wiki/MySQL_Proxy_RW_Splitting#Limitations

Oracle Corporation, MySQL-Referenzhandbuch, 16.1.3. Replication and Binary Logging Options and Variables (2010), http://dev.mysql.com/doc/refman/5.1/en/replication-options.html#option_mysqld_server-id

c) Ehrenwörtliche Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt zu haben. Wörtliche und sinngemäße Zitate sind kenntlich gemacht. Über Zitierrichtlinien bin ich schriftlich informiert worden.

Bernd Michael Helm Dohna, 26.07.2010