



Fakultät für Informatik  
Professur Rechnernetze und verteilte Systeme

# Studienarbeit

High-Availability für Zope

Marko Damaschke

Chemnitz, den 22. Dezember 2004

**Prüfer:** Prof. Dr.-Ing. habil. Uwe Hübner

**Betreuer:** Dipl.-Inf. Ronald Schmidt

**Damaschke, Marko**

High-Availability für Zope

Studienarbeit, Fakultät für Informatik

Technische Universität Chemnitz, Dezember 2004

## **Aufgabenstellung**

Im Rahmen des Projektes Bildungsmarktplatz Sachsen wird ZOPE und eine PostgreSQL-Datenbank als Applikations-Server-Struktur eingesetzt. Um den Anforderungen eines hochverfügbaren Angebots gerecht zu werden, sollen die Möglichkeiten der Lastverteilung und Mechanismen für Hochverfügbarkeit genutzt werden. Mechanismen zur Lastverteilung existieren bereits in der ZOPE-verwandten Implementierung ZEO.

Im Rahmen dieser Arbeit sollen die Anforderungen für Projekte mit ZOPE in Bezug auf Hochverfügbarkeit (High-Availability), Performance (Parallelisierung) und Caching-Strategien erarbeitet werden. Die Anforderungen hinsichtlich Caching sollen untersucht und klassifiziert, verschiedene Caching-Strategien betrachtet und verglichen sowie Empfehlungen als Resultate ausgesprochen werden.

Kostengünstige Lösungen für die Server-Struktur des Projekts Bildungsmarktplatz Sachsen sollen gefunden werden, wobei der Schwerpunkt auf Opensource-Lösungen liegt. Diese sollen statistischen Untersuchungen unterzogen und die Optimierungsmöglichkeiten von ZOPE gefunden werden.



# Inhaltsverzeichnis

<b>1</b>	<b>Aufgabenstellung und vorgefundene Gegebenheiten</b>	<b>1</b>
1.1	Ziel der Arbeit . . . . .	1
1.2	Rahmen der Arbeit . . . . .	1
1.3	Ausgangspunkt . . . . .	2
1.4	Aufbau der Arbeit . . . . .	3
<b>2</b>	<b>Grundlagenbetrachtungen</b>	<b>4</b>
2.1	Die Teile der Serverstruktur . . . . .	4
2.1.1	Der ZOPE-Server . . . . .	5
2.1.2	Die ZODB . . . . .	5
2.1.3	Plone . . . . .	5
2.1.4	Der PostgreSQL-Datenbankserver . . . . .	6
2.1.5	Der Apache-Webserver . . . . .	6
2.2	Performanz . . . . .	7
2.2.1	Was versteht man unter Performanz? . . . . .	7
2.2.2	Was gehört also in eine Performanz-Betrachtung? . . . . .	8
2.3	Was ist Lastverteilung und was ihr Ziel? . . . . .	8
2.4	Was ist Caching und dessen Ziel? . . . . .	10
2.5	Was versteht man unter einem hochverfügbarem System? . . . . .	11
<b>3</b>	<b>Die Gegebenheiten im Bildungsmarktplatz Sachsen</b>	<b>13</b>
3.1	Lastverteilung . . . . .	13
3.2	Caching . . . . .	14
3.2.1	Nutzer- und Objekt-Typen . . . . .	14
3.2.2	Betrachtungen zum Caching . . . . .	15
3.2.3	Die Objekte und Rollen im BMS . . . . .	15
3.3	Hochverfügbarkeit im BMS . . . . .	17
3.3.1	Zu betrachtende Komponenten . . . . .	18
3.4	Die Ansatzideen zum Bildungsmarktplatz . . . . .	19
<b>4</b>	<b>Lösungen und Produkte zu den genannten Problemfällen</b>	<b>20</b>
4.1	APE - erste Idee und deren Probleme . . . . .	20
4.2	Ansätze zur Lastverteilung . . . . .	21

4.2.1	Der ZEO-Server . . . . .	21
4.2.2	Dienst- und Servernamen – kein echtes Loadbalancing . . . . .	23
4.2.3	Lastverteilung mittels virtuellem Server - Switch oder LVS . . . . .	23
	Switchbasierte Lastverteilung . . . . .	23
	Lastverteilung durch LVS . . . . .	24
4.2.4	Lastverteilung mittels Round-Robin-DNS . . . . .	25
4.3	Caching-Produkte . . . . .	26
4.3.1	ZOPE-Standard-Cache-Manager . . . . .	26
4.3.2	RAM-Cache-Manager with Age . . . . .	26
4.3.3	FS-Cache-Manager . . . . .	27
4.3.4	ZOPE-Proxy-Cache-Manager . . . . .	28
4.3.5	Cache Controlled ZSQL Methods . . . . .	29
4.3.6	Die Produkte im Vergleich . . . . .	30
4.3.7	Darüber hinausgehende Entwicklungen von Caching-Produkten . . . . .	32
4.3.8	Contentbasiertes Caching im Apache . . . . .	33
4.4	Wie kann eine hohe Verfügbarkeit erreicht werden? . . . . .	33
4.4.1	Basisüberlegungen . . . . .	33
4.4.2	Zugriffsumleitung . . . . .	34
4.4.3	Rekonfiguration . . . . .	35
4.4.4	Replikation . . . . .	36
	Datenreplikation am PostgreSQL . . . . .	36
	Replikation der ZODB . . . . .	39
4.5	Überwachungsmechanismen . . . . .	40
4.5.1	heartbeat . . . . .	41
4.5.2	fake . . . . .	41
4.5.3	mon . . . . .	42
4.5.4	failover . . . . .	42
4.5.5	keepalived . . . . .	42
4.5.6	VRRP (ehemals failoverd) . . . . .	43
4.5.7	Big Brother . . . . .	43
<b>5</b>	<b>Der Bildungsmarktplatz Sachsen – eine Empfehlung zur Serverstruktur</b> <b>44</b>	
5.1	Konzept zur Lastverteilung . . . . .	44
5.2	Caching . . . . .	45
5.3	Hochverfügbarkeit . . . . .	46
5.3.1	Überwachung und IP-Übernahme . . . . .	46
5.3.2	Datenbank-Replikation . . . . .	47
	ZODB-Replikation . . . . .	47
	PSQL-Replikation . . . . .	48
5.3.3	Hinweise hinsichtlich eines Failback . . . . .	48
5.4	Mögliche Zustände der Plattform . . . . .	48

<b>6 Fazit</b>	<b>52</b>
<b>Literaturverzeichnis</b>	<b>53</b>



# 1 Aufgabenstellung und vorgefundene Gegebenheiten

## 1.1 Ziel der Arbeit

Im Rahmen dieser vorliegenden Arbeit soll untersucht werden, welche Möglichkeiten zur Sicherung einer möglichst hohen Verfügbarkeit (High-Availability), Mechanismen zur Lastverteilung mittels des ZEO-Produkts oder Ähnlichem sowie welche Strategien des Caching sinnvoll an einem ZOPE-Server zum Einsatz kommen können.

Die Arbeit untersucht dabei die Einsatzmöglichkeiten von bereits vorhandenen und die eventuelle Notwendigkeit der Eigenimplementierung weiterer Produkte der ZOPE-Entwicklung.

## 1.2 Rahmen der Arbeit

Diese Arbeit soll im Rahmen einer Studienarbeit versuchen, Empfehlungen für eine Serverstruktur und geeignete Mechanismen und Werkzeuge zu geben, die den Aufbau einer ZOPE-basierte Webapplikation zu einer hochverfügbaren und performanten Umgebung ermöglichen.

Basis und Stein des Anstosses ist die technische Realisierung der Serverstruktur des Bildungsmarktplatzes Sachsen (<http://www.bildungsmarkt-sachsen.de>). Dieses Projekt stellt eine Anbieter übergreifende Plattform für die verschiedensten (Weiter-) Bildungsangebote in Sachsen dar.

Anbieter von Bildungsangeboten sollen in die Lage versetzt werden, ihre Angebote webbasiert einzupflegen und aufzubereiten. Bildungsinteressenten hingegen will man mittels geeigneter Suchmethoden befähigen, alle Angebote zu finden, die ihren Interessen entsprechen. Die Aufbereitung muss dazu intuitiv und leicht zu bedienen sein und möglichst alle Bedürfnisse, sowohl der Suche als auch der Bereitstellung, treffen.

Daher wurde sich für eine webbasierte Anwendung entschieden, die plattformübergreifend nutzbar ist, da nur ein Webbrowser vorausgesetzt wird. Eine geschickt aufbereitete Oberfläche soll den Workflow unterstützen und sowohl unbedarfte Bildungsinteressenten, als auch Laien unter den Bildungsanbietern leiten.

## 1.3 Ausgangspunkt

Zu Beginn der Arbeit bestand die Serverstruktur aus zwei dedizierten Servern, die im Universitätsrechenzentrum der TU Chemnitz stehen und auf Basis des Betriebssystems *Fedora Core 1.0* laufen.

Als funktionale Grundlage der Anwendung wurde eine Plattform bestehend aus dem ZOPE-Applikationsserver und dem darauf aufsetzenden Content Management System Plone, einem als Proxy vorgeschaltene Apache-HTTP-Webserver sowie einer PostgreSQL-Datenbank gewählt.

Für den Einsatz des PostgreSQL wurde sich entschieden, da die Angebotsdaten einfach verfügbar und wiederverwendbar aufbewahrt werden und mit verschiedenen Werkzeugen durchsuchbar sein sollten, weshalb diese Ablage in einer relationalen Datenbank, unabhängig der objektbasierten ZOPE-eigenen ZODB, im Serversystem gewählt wurde.

Der Apache-HTTP-Server dient in der Struktur als Proxy und Cache für den ZOPE-internen Webserver.

Die Entwicklung fand auf dieser Serverstruktur primär auf einer Maschine statt und wurde aus Sicherheits- und Gründen der Ausfallüberbrückung manuell auf den zweiten Server kopiert.

Nun ist die leistungsstarke Bereitstellung der Anwendung geplant. Demnach soll es selbst im Falle eines Ausfalls einzelner Komponenten in der Struktur nicht zu einem Gesamtausfall kommen. Ist der Gesamtausfall unumgänglich, soll er möglichst kurzzeitig sein.

Erwünscht ist außerdem, dass das System selbst einer großen Zahl gleichzeitig darauf zugreifender Nutzer ein zügiges und flüssiges Arbeiten ermöglicht. Demnach ist eine minimale Reaktionszeit des Systems, mit der auf Anfragen von Nutzern reagiert wird, gewollt.

Das Gesamtserver-System besteht aus zwei identischen PC-Systemen, von denen klar war, dass sie jeweils die volle Funktionalität der Anwendung beherbergen können. Um allerdings einen möglichst guten „Funktionalitätsdurchsatz“ zu erreichen, sollen beide Rechner im Normalbetrieb parallel arbeiten. Kommt es allerdings auf einem der beiden Systeme zu einer Fehlfunktion, sollen beide Rechner jeweils in der Lage sein, die Gesamtfunktionalität zu übernehmen und aufrecht zu erhalten, bis ein Failback die Wiederaufnahme des Normalbetriebs ermöglicht. Diese Übernahme sollte automatisch und idealerweise für Nutzer, die zum Zeitpunkt des Ausfalls das System nutzen, möglichst unbemerkt erfolgen.

Es geht also primär darum zu betrachten, welche Möglichkeiten bestehen, um das System schnell, stabil und möglichst ausfallsicher zu machen - kurz das Gesamtsystem mit entsprechender Perfomanz auszustatten. Dazu sollen vor allem Open-Source-Produkte und Mechanismen betrachtet werden, die frei verfügbar sind.

## **1.4 Aufbau der Arbeit**

In den kommenden Abschnitten will ich erst kurz auf die theoretischen Hintergründe eingehen, danach schauen, welche Punkte und Nutzungsfälle im Projekt Bildungsmarktplatz Sachsen auftreten und welche Probleme und Ansatzpunkte zur Optimierung vorhanden sind. Im Anschluss versuche ich, Lösungen, Werkzeuge und Mechanismen zur Verbesserung oder Aufhebung der im vorgenannten Abschnitt aufgezeigten Schwachpunkte zu finden. Im vorletzten Abschnitt versuche ich eine Kombination, die eine bestmögliche Gesamtlösung bietet, vorzustellen.

## 2 Grundlagenbetrachtungen

Dieses Kapitel dient der Schaffung einer einheitlichen Begriffsbasis und der theoretischen Vorüberlegung.

Nach einer Betrachtung der Komponenten der Server-Software, dann einer Klärung der Begriffe Caching, Lastverteilung, Hochverfügbarkeit und Performanz, folgt der Versuch, die Grundkonzepte jeweils dazu kurz zu umreißen.

### 2.1 Die Teile der Serverstruktur

Die Komponenten der vorgefundenen Server sind der ZOPE-Applikationsserver mit seiner Speicherorganisation in Form der ZODB, die auf ZOPE aufsetzende CMS-Plattform Plone, der PostgreSQL-Datenbank-Server und der Apache-Webserver.

Einen bildlichen Eindruck soll folgende Grafik vermitteln:

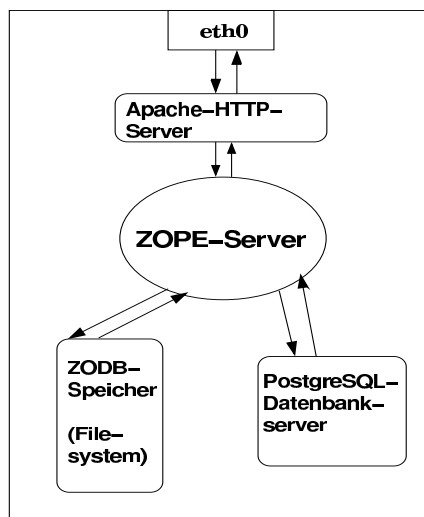


Abbildung 2.1: Grundlegende Serverstruktur

### 2.1.1 Der Zope-Server

Der auf [zop04] bereitgestellte Zope-Server ist ein Open-Source-Applikationsserver für webbasierte Anwendungen, der in der Skriptsprache Python (siehe [pyt04]) implementiert ist und durch eine freie Entwicklergemeinschaft stark weiterentwickelt wird.

Er erstellt Webanwendungen aus einzelnen Objekten (Bilder, Pagetemplates, HTML-Seiten, Python- und Perlskripten) und rendert diese dynamisch zu Webseiten, die den Workflow der Webapplikation darstellen.

Der Zope-Server wurde als zentrales Element der Webapplikation für den Bildungsmarktplatz Sachsen ausgewählt.

Im Umfeld des Zope-Applikationsservers fällt häufig der Begriff des Produkts. Da es in dieser Arbeit ebenfalls genutzt wird, möchte ich kurz erklären, wie es zu verstehen ist. Während im allgemeinen Umfeld ein Produkt ein fertiges Werkstück einer Produktion ist. Im Softwareumfeld ist es das Ergebnis einer Implementierung und Programmierung (im Gegensatz zur Lösung, die nur ein Konzept stellt), also ein fertiges Stück Software. Im Zope-Umfeld wird dieser Begriff für alle zusätzlich installierbaren und nutzbaren Software-Erweiterungen verstanden. Der Zope-Server stellt also eine Basis und Framework zur Nutzung verschiedener Erweiterungen, die funktionale Zusätze bereitstellen – den Produkten.

### 2.1.2 Die ZODB

ZODB steht für Zope Object Database und ist eine Zope-eigene objektorientierte Datenbank, in der sämtliche Inhalte, derer sich der Zope-Server bedient, abgelegt werden.

Sie ist Bestandteil der Zope-Distribution und wird im Standardfalle in einem speziellen, von Zope organisierten File im Dateisystem des Zope-Servers gehalten. Durch verschiedene Erweiterungen kann die Ablage der ZODB auch anderweitig geschehen. Beispielsweise wird mittels des APE-Produkts versucht, diese objektorientierte Datenbank auf eine relationale Datenbank abzubilden, was eine Speicherung auf den weiter verbreiteten relationalen Datenbankersystemen ermöglicht. Dies funktioniert allerdings nur mit Einschränkungen; siehe dazu 4.1.

Eine inzwischen weitverbreitete Erweiterung zur Datenablage ist der ZEO-Server (siehe 4.2.1).

### 2.1.3 Plone

Plone ist ein freies Content Management System (CMS), welches auf das Content Management Framework als Bestandteil des Zope-Systems zurückgreift.

Selbst stellt sich das Projekt auf [plo04] so vor „*Plone: A user friendly and powerful open source Content Management System*“, während sich das Zope CMF wie folgt selbst versteht „*The Content Management Framework (CMF) for Zope from Zope Corporation provides a powerful, tailorable platform for building content management applications.*“, wie auf

[cmf04] ersichtlich.

Dieses CMS soll die Erstellung und Bearbeitung durch verschiedene Personen im Rahmen eines vorgegebenen Workflows ermöglichen. Dadurch sollen die Personen, die die Information besitzen, auch die Bearbeiter der Information in der Webpräsenz sein. Intuitive Handhabung und der Besitz geringstmöglichen technischen Verständnisses sollen ausreichen, die Inhalte zu bearbeiten.

Auf den Bildungsmarktplatz Sachsen werden verschiedenste Anbieter eigenständig ihre Angebote einstellen und bearbeiten können und gleichzeitig Angebotsinteressenten eine gute Suchmöglichkeit und Präsentation der verschiedensten Angebote erfahren. Daher wurde sich für Plone entschieden.

### 2.1.4 Der PostgreSQL-Datenbankserver

Dieser Datenbankserver ist eine Open-Source-Entwicklung, die dank seiner Mächtigkeit und Stabilität auch zum produktiven Einsatz geeignet ist.

Die Entscheidung für diesen Server fiel aufgrund der recht einfachen und flexiblen BSD-Lizenz, unter der er voröfentlicht wird, und seiner Fähigkeiten, wie der Unterstützung von Transaktionen, Subselects und Stored-Procedures in der Datenbank in Python, was sich in der Python-geprägten Umgebung des ZOPE-Servers günstig darstellt.

Genauerer zu den aktuellen Entwicklungen können auf [psq04] nachgelesen werden.

### 2.1.5 Der Apache-Webserver

Der Apache-Webserver ist ebenfalls eine Open-Source-Entwicklung und einer der weitest verbreiteten Webserver, der stabil und aufgrund der großen Entwickler- und Nutzergemeinde robust implementiert ist. Seine Entwicklergemeinde findet sich unter [apa04] beheimatet. Der Webserver kommt im Rahmen des Projektes als Proxy für den ZOPE-eigenen Webserver zum Einsatz. Dies wird seitens des ZOPE-Projekts im Produktiveinsatz empfohlen, um Implementierungsschwächen des ZOPE-eigenen Webservers für mögliche Angreifer oder falsch konfigurierte Webklienten nicht nutzbar zu machen. Außerdem bieten sich dann Möglichkeiten des Cachings von komplett gerenderten Seiten und damit der Entlastung des ZOPE-Servers bei vielen Zugriffen. Ein weiteres wichtiges Augenmerk für den Einsatz stellt die einfache Einbindung von SSL-Verbindungen in den Apache dar, wodurch sich die Einrichtung der Ermöglichung von HTTPS-Verbindungen an den ZOPE-Server sehr vereinfacht. Diese wiederum ist aufgrund der Nutzerauthentifizierung wünschenswert und sinnvoll.

Alle diese Komponenten sind die grundlegenden Bausteine, aus denen die Webapplikation des Bildungsmarktplatzes Sachsen aufgebaut wird.

## 2.2 Performanz

### 2.2.1 Was versteht man unter Performanz?

**Performance** *engl.*: „Vorführung“ die; –, –s: 1. dem Happening ähnliche, meist von einem einzelnen Künstler dargebotene künstlerische Aktion. 2. prozentualer Wertzuwachs des Vermögens einer Investmentgesellschaft od. auch eines einzelnen Wertpapiers (Bankw.). 3. Leistungsniveau, –stärke eines Rechners (EDV).

Diese recht grobe lexikalische Definition aus [dD97] erfasst, wie in vielen Fällen, nur einen kleinen Ausschnitt dessen, was im fachlichen Kontext unter selbigem Begriff verstanden wird.

Daher einige weitere Definitionen hinsichtlich des Begriffes im fachlichen Umfeld. So erweitert die Sammlung von [wpp04] den fachlichen Kontext zu:

Performanz ist

1. ein Begriff aus der Sprachwissenschaft, siehe Kompetenz und Performanz
2. ein aus dem Englischen stammender Begriff für Aktionskunst, siehe Performance
3. in der Informatik das Zeitverhalten von Programmen

Die zuletzt gemachte Aussage trifft die Bedeutung entgegen der Duden-Definition deutlich besser. So ist die Leistungsfähigkeit eines Rechners durch seine Hardwarekomponenten bestimmt und kann nur durch Ausbau derer verändert werden.

Dies ist nicht Bestandteil der Betrachtungen dieser Arbeit. Nichtsdestotrotz wird versucht, das Zusammenspiel von Hard- und Software so zu beeinflussen, dass ein möglichst großer Leistungsdurchsatz im Sinne der gewünschten Funktionalität erzielt wird.

Die Performanz ist üblicherweise der funktionale Datendurchsatz eines Rechensystems in einem festen Zeitabschnitt oder die Abarbeitungszeit einer festgelegten Verarbeitungsfolge. Ziel ist es dabei natürlich, diese Datenmenge zu maximieren beziehungsweise die Zeit zu minimieren. In diesem Zusammenhang steht ebenfalls der Begriff des Speedup, welcher das relative Verhältnis des Datendurchsatzes beziehungsweise der Abarbeitungszeit vor und nach verschiedenen Optimierungs- oder Parallelisierungsmechanismen darstellt. So wird im Bereich des Clusterings, also der Parallelcomputer aus Einzelrechnern, im Speedup die Differenz der Abarbeitungszeit in der sequentiellen zur parallelen Abarbeitung dargestellt. Ähnliches findet sich im Feld der SMP-Computer, wobei dort die prozessorinternen Daten- oder Steuerpfade vervielfältigt und so das Verhältnis verschiedener Anzahlen von Pfaden im Speedup ins Verhältnis gesetzt werden.

Den Speedup und damit die Performanz-Gewinnung wird gern in einfachen Formeln der Art

$$\text{Speedup} = \text{Zeit}(\text{parallele Abarbeitung}) / \text{Zeit}(\text{sequentielle Abarbeitung}) \quad (2.1)$$

dargestellt. Ähnlich wird mein Ansatz bei der Betrachtung der Effektivität der verschiedenen Caching-Lösungen sein. Ich stelle die Abarbeitung einer bestimmten Anzahl von Anfragen durch den ZOPE-Server mit und ohne den Einsatz der einzelnen Produkte gegenüber und bestimme dann einen „Speedup“ derart:

$$\text{Speedup} = \text{Zeit(ohne Caching – Produkt)} / \text{Zeit(mit Caching – Produkt)} \quad (2.2)$$

Die vorliegende Arbeit beschäftigt sich also vorwiegend mit den Ansatzpunkten, die in der ZOPE-Applikation selbst als Einflussfaktoren vorhanden sind, um die Leistungsfähigkeit zu optimieren.

Doch einige Produkte, die in den folgenden Kapiteln besprochen werden, verlangen, aufgrund der eingesetzten Mittel, nach einem bestimmten Betriebssystem. Ebenfalls werden viele Aussagen zur Hochverfügbarkeit sich speziell an den Möglichkeiten der Serverplattform orientieren, die auf dem Bildungsmarktplatz Sachsen, der als Rahmen der Arbeit dient, zur Verfügung stehen. Dies ist das Linux-Betriebssystem, der Apache-Webserver und das PostgreSQL-Datenbank-System.

Inwieweit hier gemachte Aussagen sich auf andere Gegebenheiten abbilden lassen, ist im Einzelfall zu prüfen und sollten bei einer Entscheidungsfindung bereits bedacht werden.

### **2.2.2 Was gehört also in eine Performanz-Betrachtung?**

Da das Ziel dieser Arbeit ein hochverfügbares Applikationsserver-System sein soll, welches gleichzeitig möglichst aktuell und schnell ein webbasiertes Angebot bereitstellen soll, werden die Möglichkeiten betrachtet, die ein sinnvolles Caching zur Beschleunigung des Systems beitragen kann, ebenso die optimale Ausnutzung der bereitgestellten Serverkapazitäten durch Lastverteilung und Parallelisierung. Die Stabilität und Ausfallsicherheit muss dann gesondert auf Möglichkeiten des Einsatzes von Hochverfügbarkeits-Konzepten untersucht werden.

## **2.3 Was ist Lastverteilung und was ihr Ziel?**

Unter dem Begriff des Loadbalancing werden alle Maßnahmen zusammenfasst, die eingesetzt werden, um über mehrere Server die Aufgaben so zu verteilen, dass die Ressourcen der Server gleichmäßig genutzt werden. Die betrachteten Ressourcen dabei sind beispielhaft die Prozessor-, Hauptspeicher- und Netzinterfaceauslastung. Es geht um das Verhältnis von maximaler Leistungsfähigkeit der einzelnen Komponente zu aktuellem Nutzungsverhalten. Dies gilt gleichermaßen für Rechensysteme mit redundanten Ressourcen, wie Mehrprozessormaschinen oder Systemen mit Mehrweg-Datenspeichern, in denen über die vorhandenen Ressourcen eine gute Verteilung der Aufgaben stattfinden soll.

Ziel dabei ist, die eingebrachten Bauteile möglichst gleichmäßig zu nutzen oder den Durchsatz zu maximieren und dabei alle Datenflusswege so auszunutzen, dass die Wartezeiten gleichmäßig und minimal werden.

Inhaltlich adäquate Definitionen finden sich bei [sni04a] beziehungsweise ebenfalls im Wissensschatz von [wpl04]:

**load balancing**

The adjustment of system and/or application components and data so that application I/O or computational demands are spread as evenly as possible across a system's physical resources. I/O load balancing may be done manually (by a human) or automatically (by some means that does not require human intervention).

**Lastverteilung**

Mit Lastverteilung (engl. load balancing) werden Verfahren beschrieben, um bei Computern umfangreiche Berechnungen oder große Mengen von Anfragen auf mehrere parallel arbeitende Systeme zu verteilen.

Dies kann sehr unterschiedliche Ausprägungen haben. Eine einfache Lastverteilung findet zum Beispiel auf Rechnern mit mehreren Prozessoren statt. Jeder Prozess kann auf einem eigenen Prozessor ausgeführt werden. Die Art der Verteilung der Prozesse auf Prozessoren kann dabei einen großen Einfluss auf die Gesamtperformance des Systems haben, da z.B. der Cache-Inhalt lokal für jeden Prozessor ist.

Ein anderes Verfahren findet man in Computerclustern. Hierbei bilden mehrere Rechner einen Verbund, der sich nach außen wie ein einzelnes System verhält. Einige mögliche Verfahren sind das Vorschalten eines Computers, der die Anfragen aufteilt oder die Verwendung von Round Robin DNS.

Lastverteilung findet auch bei großen Server-Farmen statt, die z.B. der Beantwortung von HTTP-Anfragen dienen. Dort sind Systeme vorgeschaltet, die nach festgelegten Kriterien die einzelnen Anfragen auf die Backend-Server verteilen. Dabei können zusätzliche Informationen aus dem HTTP-Request verwendet werden, um alle zu einer Session mit einem Benutzer gehörenden Pakete an den gleichen Server zu schicken. Dies ist auch bei der Nutzung von SSL zur Verschlüsselung der Kommunikation wichtig, damit nicht für jede Anfrage ein neuer SSL-Handshake durchgeführt werden muss.

Eine gute Umsetzung einer Lastverteilung erfordert immer Informationen darüber, wie die Auslastung der Zielsysteme aussieht.

Problem beim Einsatz ist natürlich einerseits, dass die Entscheidung der Verteilung möglichst keine Wartezeit erzeugt und die Folge der Entscheidung erst im Nachhinein, bei Bekanntwerden der verursachten Kosten, bekannt wird. Es muss also aufgrund stochastischer Werte entschieden werden. Außerdem soll die Ermittlung der Entscheidungsgrundlage nicht mehr Kosten oder Last verursachen, als die Bearbeitung der eigentlichen Aufgabe, die zu verteilen ist.

Üblicherweise werden bei Lastverteilungen im Netzverkehrbereich zur Entscheidungsfindung ausschließlich die Aspekte der Netzlast betrachtet. Die Last von anderen Systemressourcen wird dabei, wenn überhaupt, meist nur in der Veränderung von Antwortzeiten oder Durchsatz eines Netzinterfaces berücksichtigt. Das spart die Kosten für die Überwachung weiterer Systemkomponenten und vereinfacht die Entscheidungsfunktion beträchtlich.

## 2.4 Was ist Caching und dessen Ziel?

Sucht man nach einer Definition des Begriffes Caching oder Cache wird man in verschiedensten Ausprägungen fündig. So beschreibt [sni04b] Cache wie folgt:

**cache**

A high speed memory or storage device used to reduce the effective time required to read data from or write data to a lower speed memory or device. Read cache holds data in anticipation that it will be requested by a client. Write cache holds data written by a client until it can be safely stored on more permanent storage media such as disk or tape.

[mtc04] vereinfacht den Aspekt auf die Betrachtungen des HTTP und definiert so:

**caching**

The storage of Web files for later re-use at a point more quickly accessed by the end user.

*Information*

Caching can happen at many places, including proxies (i.e. the user's ISP) and the user's local machine. The objective is to make efficient use of resources and speed the delivery of content to the end user.

While caching can have a positive impact on the user's experience, it can have a negative impact for site publishers, resulting in undercounts of page views and ad impressions. In response to this problem, sites have implemented various cache-busting techniques to better ensure that all performance statistics are accurately measured.

Als Netzanbieter lexikalischer Definitionen von Begriffen im technischen Umfeld definiert [tec04] den Begriff recht umfangreich wie folgt:

**cache**

A cache (pronounced CASH) is a place to store something temporarily. The files you automatically request by looking at a Web page are stored on your hard disk in a cache subdirectory under the directory for your browser (for example, Internet Explorer). When you return to a page you've recently looked at, the browser can get it from the cache rather than the original server, saving you time and the network the burden of some additional traffic. You can usually vary the size of your cache, depending on your particular browser.

Computers include caches at several levels of operation, including cache memory and a disk cache. Caching can also be implemented for Internet content by distributing it to multiple servers that are periodically refreshed. (The use of the term in this context is closely related to the general concept of a distributed information base.)

Eine letzte, der so vielseitig ausgeprägten Definitionen des Caches sei [wpc04] zitiert:

Cache bezeichnet in der EDV einen besonders schnellen Speicher, der bei CPUs meist direkt auf dem Prozessor-Die integriert ist. Bei Festplatten liegt er auf der Steuerplatine. In ihm werden Daten und Programminstruktionen zwischengelagert, so dass diese bei einem Zugriff durch die CPU schnell zur Bearbeitung bereitstehen. Ziel des Cache-Speichers ist die Verringerung der Anzahl der Speicherzugriffe.

Den Namen verdankt der Cache der Tatsache, dass er als Teil des Steuerwerks im Verborgenen arbeitet. Nur selten spricht ein Programmierer den Cache explizit an, etwa um ihn zu leeren (Flush). Wörtlich aus dem Englischen übersetzt bedeutet Cache (entlehnt aus dem französischen cacher - verbergen, caché - verborgen) soviel wie 'geheimes Lager'. Aus den Quotes-of-the-day eines Linux Betriebssystems: 'Cache: A very expensive part of the memory system of a computer that no one is supposed to know is there.'

Das Funktionieren des Caches beruht auf der so genannten Lokalitätseigenschaft.

Spricht man vom Caching, geht es also um die Speicherung bestimmter Inhalte oder Zwischenergebnisse auf dem Pfad von der erzeugenden Instanz zum endgültigen Nutzer der Daten. Allgemein geht es um die Verminderung aufwendiger Berechnungen oder teurer Übertragungen. Aber eben auch zur Entlastung der Flaschenhalse in den Datenpfaden kommt Caching zum Einsatz, indem schnelle jedoch kleinere und deutlich teurere Speichereinheiten vor langsameren zum Einsatz kommen, um eine Beschleunigung des Zugriffs auf die langsamen, aber größeren Speicher zu erreichen.

## 2.5 Was versteht man unter einem hochverfügbarem System?

Mit einem hochverfügbarem System wird gemeinhin ein Rechnersystem verbunden, welches die ihm angedachte Aufgabe mit einer möglichst hohen Zuverlässigkeit erfüllt.

[MB01] definiert den Begriff folgendermaßen:

Hochverfügbarkeit beschreibt ein System, das in der Lage ist, Fehler zu reduzieren oder mit ihnen umzugehen und ebenso geplante Stillstandszeiten zu minimieren. Ein System gilt als hochverfügbar, wenn es ohne Eingriff von außen eine Anwendung wieder startet und keine oder eine kurze Unterbrechung wahrnehmbar ist.

Ähnlich sieht [sni04c] die Hochverfügbarkeit, im Englischen „High availability“, wie folgt eingeordnet:

The ability of a system to perform its function continuously (without interruption) for a significantly longer period of time than the reliabilities of its individual components would suggest. High availability is most often achieved through failure tolerance. High availability is not an easily quantifiable term. Both the bounds of a system that is called highly available and the degree to which its availability is extraordinary must be clearly understood on a case-by-case basis.

Es geht also darum, ein System oder eine Anwendung auch dann (wieder) verfügbar zu machen, wenn entweder ein Fehler auftrat, der den korrekten Ablauf beeinflusste, aber auch wenn ein geplantes Wartungsfenster beispielsweise zur Abschaltung von Teilkomponenten führt. Dazu werden üblicherweise die sensitiven Komponenten doppelt beziehungsweise redundant vorgehalten, um im Falle eines Ausfalls, die Funktionalität der ausgefallenen Komponente durch einen gleichwertigen Ersatz aufrechterhalten zu lassen.

Um die in der Definition als kurze Ausfallzeiten benannten Größenordnungen an einem Zahlenbeispiel vorzustellen, sei erwähnt, dass bei hochverfügbaren Systemen oft von Verfügbarkeiten von mehr als 99 Prozent ausgegangen wird. Dabei ist noch zu unterscheiden, in welchem zeitlichen Vergleich dieser Prozentsatz zu erreichen ist - pro Tag, Woche, Monat oder Jahr.

Nimmt man beispielsweise eine Rate von 99,5% pro Monat als Grundlage, lässt dies einen Ausfall von 216 Minuten oder circa 3,5 Stunden pro Monat zu. Ist dieser Prozentsatz hingegen pro Jahr anzulegen, kann es innerhalb eines Jahres zu einem Gesamtausfall von circa 43,5 Stunden kommen, also auch etwa 1,5 Tage am Stück, wenn die restliche Zeit ausfallfrei bleibt. Daher werden an sehr kritische Systeme, etwa in Kernkraftwerken, sehr hohe Prozentsätze an Funktionalität, deutlich über 99,9%, gestellt und diese in sehr kleine zeitliche Kontexte gesetzt.

Da von vornherein gewünscht war, die Serverplattform des Bildungsmarktplatzes Sachsen ausfallsicher oder wenigstens mit hoher Verfügbarkeit bereitzustellen, wurden zwei identische Serversysteme beschafft, die geeignet verknüpft, diesem Umstand Rechnung tragen sollen.

Weiterhin soll der Ausfall respektive die Ersetzung einzelner Komponenten für den Nutzer transparent oder unbemerkt erfolgen und möglichst wenige Ergebnisse der bisherigen Arbeit sollten verloren gehen.

## 3 Die Gegebenheiten im Bildungsmarktplatz Sachsen

In diesem Kapitel will ich auf die Gegebenheiten sowohl hardware-, als auch softwareseitig eingehen. Außerdem will ich die bestehenden Ideen aufzeigen, die Ausgangspunkt der Überlegungen und Arbeit waren.

Wie bereits im vorhergehenden Kapitel gehe ich dabei auf die 3 Punkte Lastverteilung, Caching und Hochverfügbarkeit ein.

### 3.1 Lastverteilung

Da die Plattform des Bildungsmarktplatzes Sachsen auf zwei Server aufsetzte und eine große Zahl von Zugriffen erwartet wird, sollte der zweite Server nicht nur als Fallback-Lösung bereitstehen, falls auf dem primären Server ein Problem auftritt oder dieser für Wartungszwecke aus dem Produktivbetrieb gelöst werden muss, sondern beide Rechner sollen aktiv am Betrieb im Normalfall beteiligt sein.

Dabei hatte bereits der Gedanke der Performanz-Steigerung starken Einfluss auf die Entscheidung. Durch die Parallelisierung der Server wird üblicherweise eine Verringerung der Antwortzeiten auf die Anfragen erreicht, was einem Speedup größer 1 nach Formel 2.1 entspricht.

Im „Normalbetrieb“ war nun also vorgesehen, dass beide Server Frontend-Funktionalität übernehmen und einen ZOPE-Server bereitstellen. Über diese Instanzen sollte dann eine geeignete Lastverteilung stattfinden, um eine optimale Ausnutzung der Ressourcen zu erreichen. Einer der Server musste dabei zusätzlich die Datenbank hosten, über deren sinnvolle und funktionale Sicherung hinsichtlich eines Ausfalls noch Klärungsbedarf bestand.

Weiterhin war gewollt, dass beide Server jeweils die Wartungszeiten des anderen überbrücken können und für diese Frist die Arbeit des Gesamtsystems übernehmen.

Als Stichwort in diesem Zusammenhang fiel der Begriff ZEO als Produkt, welches dahingehend Unterstützung leisten sollte, mehrere ZOPE-Instanzen lastverteilt oder überhaupt verteilt zu betreiben. Die eigentliche Funktion und deren Umfang stand zur Klärung an.

Die erste Idee war, beide Maschinen parallel aufzusetzen und darüber eine Lastverteilung anzusetzen. Unklar war, wie ein Abgleich der beiden ZODBs stattfinden konnte, um einen einheitlichen Datenstand, vor allem hinsichtlich der Nutzerdaten zu gewähren.

Die Struktur hinsichtlich der DNS-Namen und IP-Nummern bestand zu Beginn der Arbeit aus den beiden Maschinennamen *clara.hrz.tu-chemnitz.de* und *richard.hrz.tu-chemnitz.de* sowie den zugehörigen IPs und dem Namen *www.bildungsmarkt-sachsen.de*, der als Alias für einen der beiden Maschinennamen eingerichtet war.

## 3.2 Caching

Prinzipiell stellt sich als erstes bei diesem Thema die Frage, was kann überhaupt sinnvoll an welcher Stelle zwischengespeichert werden und mit welchem Erfolg, Aufwand und Nutzen ist das Caching an dieser Stelle verbunden.

Prinzipiell war zum Anfang der Arbeit klar, dass natürlich ein HTTP-Caching möglich ist. Dabei war zu klären, inwieweit ist dies steuerbar und welche Inhalte lohnten überhaupt ein derartiges Caching, da ja viele dynamische Webseiten erzeugt werden sollten, die sich inhaltlich von Nutzer zu Nutzer oder von Besuch zu Besuch oder aber von Suchanfrage zu Suchanfrage unterscheiden. Unklar war, ob ein derartiges Caching ständige Cache-Verfehlungen oder die Auslieferung veralteter Seiten verursachte.

Weiterhin bestand Ungewissheit in Hinblick auf die verschiedenen Cache-Produkte, die es direkt für ZOPE gab, welche Funktion sie versehen, welche sinnvolle Nutzung denkbar wäre. So stand beispielsweise im Raum, ob das individuelle Cachen von einzelnen Elementen eines Pagetemplates möglich ist, da viele Seiten des Bildungsmarktplatzes ähnliche Objekte von Pagetemplates enthalten, aber der Inhalt aus einer Datenbankabfrage generiert wird.

### 3.2.1 Nutzer- und Objekt-Typen

ZOPE vergibt verschiedene Nutzerrollen, die beim Caching für verschiedene Strategien benutzt werden sollten. Primär dienen diese Rollen der Rechteverteilung auf verschiedenen Objekten innerhalb der Objekthierarchie. Ursprünglich gibt es im ZOPE drei Rollen – der Objektbesitzer, der Manager und der anonyme Nutzer.

Der Manager dient der Verwaltung und Bearbeitung einzelner Objekte oder Objekt-Teilhierarchien. Im Rahmen des Projektes werden die inhaltlichen Bearbeiter und Anbieter wahrscheinlich mit dieser Rolle in ihrer Teilhierarchie arbeiten.

Der Objektbesitzer ist der Eigentümer eines Objekts, dem alle Rechte an dem einzelnen Objekt zustehen – er dient als letzter Administrator, falls ein Manager seiner Rolle und damit allen Managern das Recht zur Administration des Objekts entzogen hat, da dem Objektbesitzer die Rechte nicht entzogen werden können.

Die dritte Rolle wird allen restlichen Nutzern, die also in der aktuellen Hierarchie (-stufe) keine anderen Rollen haben, zugewiesen.

Da der Bearbeiter natürlich die Ergebnisse seines Schaffens sofort sehen möchte, sobald er die Veränderungen vorgenommen hat, muss ihm eine andere Caching-Strategie zu teil werden, als beispielsweise einem anonymen Betrachter.

Weiterhin gibt es verschiedene Objekte, die wiederum unterschiedlich behandelt werden sollten. Bedenkt man beispielsweise, dass sich Bilder selten ändern, Inhalte von Webseiten, die mittels Skripten aus Datenbankanfragen generiert werden und etwa tagesaktuelle Neuigkeiten oder Angebote präsentieren, dagegen häufig, kann man Bildern und statischen Textseiten längere Aufbewahrungszeiten in Caches gewähren, als Skripten oder Pagetemplates, die dynamisch Inhalte erzeugen.

Bei dem vorliegenden System geht es bei der Betrachtung des Cachings vor allem um das aufwendige Rendering der Objekte aus der ZODB zu einer Webseite unter der Annahme einer gleichzeitigen Nutzung des Systems von vielen Betrachtern. Außerdem soll, wie bei vielen vernetzten Systemen, der Aufwand für die Datenübertragung gering gehalten werden. Dies steht dabei im Konflikt mit der Aktualität der angezeigten Inhalte, der Bearbeitung direkt am Netz und der sofortigen Sichtbarkeit der Neuerungen, und einem ausgewogenem Kostenverhältnis zwischen der Aktualität und der Übertragung bezogen auf bestimmte Nutzergruppen.

### **3.2.2 Betrachtungen zum Caching**

Prinzipiell finden sich verschiedenste Eingriffspunkte, um Informationen, Objekte und Dokumente zwischenspeichern und somit den erneuten Zugriff auf diese zu beschleunigen. Das Serverszenario des Projektes „Bildungsmarktplatz Sachsen“ sieht vor, dass das ZOPE eingebettet liegt zwischen einer PostgreSQL-Datenbank und einem Apache-Webserver, der als externer Cache fungiert. Ausgehend davon ist ein Caching sowohl im Apache oder auch weiteren HTTP-Caches auf dem Weg durchs Netz zum Nutzer, gesteuert durch den ZOPE-Server, eine Zwischenspeicherung verschiedener SQL-Anfrage-Ergebnisse, sowie verschiedenste Cachingstrategien im ZOPE selbst denkbar.

### **3.2.3 Die Objekte und Rollen im BMS**

In der Struktur des Bildungsmarktplatzes Sachsen ist vorgesehen, dass die Bildungsanbieter sich anmelden müssen, um ihre Angebote einzustellen oder zu bearbeiten. Dabei werden verschiedenste Rollen genutzt, die der Abbildung des Workflow einerseits beim Bildungsanbieter, andererseits beim Bildungsmarktplatz gerecht werden. So gibt es unter anderem Anbieter, Reviewer und Manager. Genauere Erklärungen zu dieser Struktur entnehmen Sie bitte Tabelle 3.1.

Weiterhin ist in einem späteren Ausbau des Projekts angedacht, dass sich der Bildungsinteressent ebenfalls anmeldet, um ihm die Möglichkeit des Angebots von so genannten Mehr-

Rolle	zugeordnete BMS-Rolle	Nutzer
Anbieter		Ein Anbieter von Bildungsangeboten, der diese im BMS einstellt.
Anonymous		Nutzer, die nicht angemeldet sind oder sonst einer Rolle zugeordnet werden können
Authenticated		angemeldete Nutzer, die in dieser Objekthierarchie keiner anderen Rolle zugeordnet werden
Manager	Admins	Administrator der gesamten BMS-Struktur
Member	(später Interessent)	angemeldeter Nutzer des BMS-Angebots
Owner		ausschließliche ZOPE-Rolle, die zur Reparatur von Rechten dient
Reviewer	CMS-Redakteur	diese Rolle berechtigt zur Pflege und Freigabe von CMS-Inhalten im BMS
Revisor	Angebots-Redakteur	Nutzer mit dieser Rolle haben die Aufgabe, von Anbietern eingestellte Angebote zu prüfen und freizugeben.

Tabelle 3.1: Nutzerrollen im BMS

wertdiensten zu bieten und Nutzerprofile erstellen zu können.

Die Objekte, aus denen sich der Auftritt des Bildungsmarktplatzes Sachsen zusammensetzt, sind vor allem Pagetemplates und Skripte sowie Skin-Vorlagen im Plone.

Inhalte, die sich lohnen außerhalb des ZOPEs zwischenspeichern, sind dabei die Portalseiten, die statisch sind und einen allgemeinen Überblick geben. Weiterhin sind die Inhalte der Skins, die sich auf jeder Seite wieder finden, von Interesse für das Caching, da diese Elemente nicht mit jeder Seite neu generiert und vor allem übertragen werden müssen. Diese Inhalte, die mit den Mitteln des HTTP-Cachings zwischengespeichert werden, sind ja vor allem dahingehend interessant, dass dem Nutzer nicht jedes Seitenelement wieder und wieder übertragen werden muss.

ZOPE-intern sind sowohl die fertig gerenderten Webseiten als auch Teile derer und das Vorhalten von oft wiederkehrenden Datenbankabfrage-Ergebnissen interessant zur Zwischenspeicherung. Es ergeben sich dadurch hoffentlich schnellere Antwortzeiten des ZOPE-Servers und geringere Datenbankverbindungen.

Leider lassen sich im ZOPE nicht alle Objekte an ein Cache-Manager-Objekt binden und somit viele Sachen nicht mit den ZOPE-eigenen Mitteln cachen. Gerade im Plone lassen sich nur Teile der Skins assoziieren. Dadurch ergibt sich die recht kleine Matrix Tabelle 3.2 an

<b>Objekttyp</b>	<b>Nutzerrolle</b>	<b>Anforderung an die Cacheaktualität</b>
einige wenige Seiten (Suchformulare) eines Plone-Portals	Manager, Reviewer	ungecached, da Änderungen nur bei CMS-Anpassungen, die dann sofort sichtbar sein sollen
	andere Rollen	cachebar, da Änderungen selten und Seiten damit recht statisch
Skinbestandteile (Bilder, css-Dateien, Internationalisierung-Sprachdateien) eines Plone-Portals	Manager, Reviewer	ungecached, da Änderungen nur bei CMS-Anpassungen, die dann sofort sichtbar sein sollen
	andere Rollen	cachebar, da Änderungen selten und Seiten damit recht statisch

Tabelle 3.2: Objekttypen im BMS

cachebaren Objekten.

Eigentlich lässt sich schon an Hand der Tabelle erkennen, dass nur recht wenige Punkte zum Cachen festzuhalten sind und zusammenfassend kann man sagen, dass die Bemühungen dahingehend laufen sollten, statische Bestandteile und Elemente der verschiedensten Seiten, eben vor allem Bilder und Stylesheets, cachebar vorzuhalten und nach Möglichkeiten zu schauen, innerhalb des Rendering-Prozesses Optimierungen und cachebare Elemente zu finden, etwa Datenbank-Anfragen oder ganze Seiten mit wiederkehrenden Randbedingungen.

### 3.3 Hochverfügbarkeit im BMS

Bei der Betrachtung der Hochverfügbarkeit muss man stets die Hard- und Softwaregegebenheiten betrachten. Natürlich ist die Funktionalität der Hardware Voraussetzung und notwendige Bedingung für das Funktionieren der Softwarekomponenten. Da sich die hardwareseitigen Methoden zur Steigerung der Verfügbarkeit auf redundantes Auslegen der Komponenten und den Einsatz von Watchdogs mehr oder minder beschränken und diese Arbeit sich auf die softwareseitigen Möglichkeiten konzentriert, werden diese Punkte hier nicht näher betrachtet. Weiterhin zieht der Ausfall einer notwendigen Hardwarekomponente den Ausfall einer Softwarekomponente nach sich, weshalb die hier genannten Mittel zur Aufrechterhaltung der Funktionalität der Softwarekomponente ebenfalls Wirkung zeigen, um den Ausfall der Hardware abzufangen.

### 3.3.1 Zu betrachtende Komponenten

Zur Klärung der Frage, welche Komponenten hinsichtlich einer Hochverfügbarkeit zu betrachten sind, ist es vorerst wichtig, sich der notwendigen Bedingungen für den Einsatz der einzelnen Systemkomponenten bewusst zu werden.

Klar im Vordergrund steht hier der ZOPE-Applikationsserver, der das zentrale Element darstellt. Dessen Ausfall kann lokal natürlich nur durch den Versuch, den ausgefallenen Prozess neuzustarten, kompensiert werden. Im Gesamtaufbau der Plattform ist dagegen eine vorübergehende Absicherung der Funktionalität durch Übernahme der Aufgaben durch den zweiten Server möglich. Doch damit der ZOPE-Server jeweils überhaupt seine Funktion entfalten kann, sind ja weitere Komponenten des Serversystems notwendig.

Als nächster wichtiger Punkt ist hier die ZODB zu nennen. Diese liegt zwar im Filesystem des ZOPE-Servers, ist aber eben ein zentrales Element. Filesysteme können Fehler aufweisen und vor allem Fehler, die durch die Hardware des Speichermediums entstehen, sind nicht selten. Darüber hinaus ist aus den Punkten der Lastverteilung zu überlegen, wie eine Verteilung der ZODB oder ein Abgleich mehrerer ZOPE-Instanzen erfolgen kann. Bei Verfahren zur Duplizierung von Daten oder bei Verteilungen können Raise-Conditions auftreten; üblicherweise ist die Frage, welchen Zustand ein System einnimmt, wenn es bei der Duplikation unterbrochen wird.

Als weiterer zentraler Datenspeicher muss ebenfalls dem Datenbank-Serversystem Aufmerksamkeit bei dem Bedenken eines möglichen Einzelausfalls zukommen. Auch hier stehen die Fragen, was passiert, wenn der Server ausfällt beispielsweise aufgrund von Fehlern in der Datenablage.

Und auch hier ist zu überlegen, wenn das Gesamtsystem redundant ausgelegt ist, ob und wie dieser Serverprozess verteilt, überwacht und gegebenenfalls auf den zweiten Server migriert werden kann. Damit stehen die bereits angesprochenen Probleme bei Abgleich oder Duplikation erneut in der Aufmerksamkeit.

Weiterhin muss der Apache-Webserver betrachtet werden, der erst den Zugriff auf die ZOPE-Instanz von außen gewährt. Fällt dieser aus, kann selbst ein intakter ZOPE-Server auf dem Rechner für den Endnutzer keine nützlichen Dienste mehr vollbringen.

Natürlich darf in der Gesamtbetrachtung nicht vergessen werden, auch das Gesamtsystem zu sehen. So können nicht nur einzelne Komponenten oder Teile der Softwarelösung eines Rechners ausfallen, sondern auch der gesamte Server. Beispielsweise durch Fehler im Betriebssystem, in Treibern oder der Hardware.

Auch kann der Ausfall einer einzelnen Komponente die völlige Erreichbarkeit der Dienste unterbinden. Genannt sei hier beispielsweise die Netzwerkkarte, deren Ausfall den Server für eine Dienstnutzung ebenso unbrauchbar macht wie der Gesamtsystemausfall. Also muss

auch der Ausfall des Gesamtsystems betrachtet werden und Möglichkeiten, diesen zu kompensieren.

### 3.4 Die Ansatzideen zum Bildungsmarktplatz

Grundlegend ist die Frage, wie man die beiden Server miteinander verbindet, um von beiden Maschinen einen gleichwertigen Dienst anbieten zu können. Dabei ist erst zu betrachten, ob und welche Abhängigkeiten zwischen einzelnen Diensten bestehen, und dann welche Mechanismen zur Abstimmung zwischen den Anwendungen der einzelnen Rechner vorhanden sind.

Die Abhängigkeiten in der Struktur des „Bildungsmarktplatzes Sachsen“ sieht nun vor, dass der Dienst erreichbar ist, wenn der Apache-Webserver an das Netzwerkinterface gebunden ist und den ZOPE-eigenen Webserver erreicht, der ZOPE-Server die ZODB und, da in Skripten darauf zugegriffen wird, die PostgreSQL-Datenbank erreichen.

Es ergibt sich schließlich grundlegend folgender Abhängigkeitsgraph zwischen den Software-Komponenten:

- *Netzwerkinterface* → *Apache HTTP Server*
- *Netzwerkinterface* → *PostgreSQL – Server*
- *Netzwerkinterface* → *ZOPE* ← *ZODB* ← *Filesystem*

Dabei sind hinsichtlich der Datenbackends, also der beiden Datenbanken, die Möglichkeiten zur Bereitstellung einheitlicher Datenstände zu überprüfen.

Bei der PostgreSQL bestünde wahrscheinlich mit einer deutlichen Steigerung der Kosten und einer weiteren Hardwarebeschaffung die Möglichkeit über ein externes Speichermedium (zum Beispiel: ein SCSI-RAID), welches von beiden Rechnern erreichbar ist, eine einheitliche Datensicht auf beiden Maschinen zu erreichen. Da auf die ZODB üblicherweise mittels eines Filehandles zugegriffen wird, welches weiteren Zugriff lockt, käme diese Variante für die ZODB bereits nicht in Betracht. Es bestand also grundlegend die Überlegung mittels geeigneter Mechanismen die beiden einzelnen ZOPE-Instanzen auf den beiden Servern zu verknüpfen und dann über beide Server eine Lastverteilung zu machen.

## 4 Lösungen und Produkte zu den genannten Problemfällen

### 4.1 APE - erste Idee und deren Probleme

APE ist ein Produkt, welches genau für die Zope-Umgebung entwickelt wurde und dazu dient, Python-Objekte derart anzupassen, dass diese in faktisch jedem Datenspeicher abgelegt werden können.

Auf [ape04] findet sich diese Erklärung:

Ape, the Adaptable Persistence Engine, adapts Python objects to storage on the filesystem, in relational databases, or practically anywhere else. It combines the advantages of transparent object persistence with arbitrary databases and formats. It is especially designed to work with ZODB, but can work independently of both Zope and ZODB.

Es ist also dazu gedacht, die Inhalte der ZODB aus dem Dateisystem, wo sie standardmäßig abgelegt ist, in andere Speichersysteme abzulegen. Allen voran relationale Datenbanken.

Die erste Idee zum Einsatz des APE-Produkts bestand darin, die beiden Zope-Server mittels APE an eine gemeinsame PostgreSQL-Datenbank anzubinden und damit ein einheitliches Datenbackend zu haben.

Diese Idee ließ sich leider nicht umsetzen, da das APE die angeschlossenen Instanzen nicht über Veränderungen an den Objekten informiert. Das bedeutet, wenn einer der Zope-Server ein Objekt in der Hierarchie anlegt, löscht oder verändert, muss die zweite Instanz benachrichtigt werden. Dies ist notwendig, da die Hierarchie einmalig beim Start eingelesen, danach zum Teil in einer temporären ZODB im Speicher gehalten wird und nur noch Veränderungen geschrieben beziehungsweise aufgrund von Signalisierungen nachgelesen werden.

Weiterhin ergaben die Tests mit dem APE-System dahingehend, dass man nicht den zentralen Speicherplatz der ZODB direkt an das APE anschliessen kann, um alle Objekte abulegen. Die Konfiguration ermöglichte nur das Erstellen eines weiteren Mount-Points für die ZODB, der dann einzelnen Teilhierarchien zugeordnet werden kann, auf dass die Objekte dieser Hierarchie dann über den Mount-Point an das APE vermittelt und somit beispielsweise in einer Datenbank abgelegt werden.

Speziell in Kombination mit dem Plone-Produkt zeigten sich allerdings weitere Probleme, wobei bestimmte Teile einer Plone-Portal-Struktur nicht serialisierbar scheinen und somit

das APE keine funktionierende Umwandlung zur Speicherung in einer relationalen Datenbank erstellen konnte.

In der späteren Folge wurde nach Untersuchung der ZEO-Erweiterung erneut das APE in Betracht gezogen, um den Datenspeicher des ZEO in eine relationale Datenbank zu verlagern. Auch dies war nicht befriedigend möglich. Einerseits war es nicht in allen Fällen möglich, den Datenspeicher eines ZEO-Servers einfach an das APE-Produkt zu binden, da die Vorgehensweise, die im APE dokumentiert ist, um eine einzelne Instanz anzubinden, zu Fehlermeldungen beim Start des ZSS führt.

Nach Tests mit verschiedenen Konfigurationen gelang die Anbindung, führte aber zu Problemen, da offenbar die Objekt-IDs, die zur Bezeichnung der Objekte zwischen ZOPE und ZODB-Adapter genutzt werden, eine andere Form haben, als die, die zwischen ZODB-Adapter und ZSS zum Einsatz kommen. So war es nicht möglich, ein zufriedenstellendes Ergebnis mit der Kombination  $ZOPE \rightarrow ZEO \rightarrow APE \rightarrow PSQL$  zu erreichen.

## 4.2 Ansätze zur Lastverteilung

Bei der Untersuchung hinsichtlich einer geeigneten Lastverteilung für den Bildungsmarktplatz Sachsen wurde erst einmal das ZEO-Produkt untersucht, welches zwischenzeitlich als eine Erweiterung der ZOPE-Distribution beiliegt.

### 4.2.1 Der ZEO-Server

Ausgesprochen bedeutet ZEO „ZOPE Enterprise Object“ und entgegen den Erwartungen war es keine Erweiterung zur Verknüpfung und Lastverteilung von ZOPE-Instanzen im Sinne eines Proxys, der die Zugriffe auf die einzelnen Instanzen koordiniert.

Hingegen ermöglicht es diese Erweiterung, mehrere Instanzen von ZOPE-Servern auf eine ZODB zugreifen zu lassen, wodurch alle Instanzen jederzeit einen einheitlichen Blick auf die Objekte hat. Dabei übernimmt der ZEO-Server ebenfalls die Benachrichtigung der einzelnen Instanzen über Veränderungen an den Objekten, der Objekthierarchie oder notwendigen Einflüssen auf die lokalen Caches der einzelnen Instanzen.

Dazu müssen alle Instanzen in der selben Version vorliegen und mit den selben Erweiterungsprodukten ausgestattet sein. Außerdem ist zu beachten, dass zwischen den ZOPE-Servern und dem ZEO-Server keine Verschlüsselung stattfindet, so dass, ist dies gewünscht, derartige Sicherungen mittels Tunnel, etwa stunnel, openssl-Tunnel oder VPN-Tunnel, erfolgen müssen.

Die einzelne ZOPE-Instanz greift nun nicht mehr auf eine ZODB im eigenen Filesystem zu, sondern nutzt eine auf einem ZSS, einem ZEO Storage Server. Dieser ist ein speziell konfigurierter ZOPE-Server, der die ZODB organisiert und die Zugriffe auf die Objekte durch die

einzelnen Instanzen verwaltet.

ZEO erlaubt ebenfalls die Kaskadierung mehrerer ZEO-Server, da Schreibzugriffe einer einzelnen Instanz bis zu deren Ausführung entweder den ZEO-Server oder die einzelne ZOPE-Instanz blockieren kann. Daher kann man dann mehrere ZSS aufsetzen, die dadurch jeweils weniger ZOPE-Instanzen bedienen und die Vermittlung der Schreiboperationen an den zentralen ZEO-Server übernehmen, der immer noch vorhanden sein muss, und den aktuell gültigen Datenstand repräsentiert. Zusätzlich ist zu erwähnen, dass eine Sensibilität gegenüber Schreiboperationen dahingehend besteht, dass bei gleichzeitigem Zugriff zweier Instanzen auf ein ZODB-Objekt ein ConflictError seitens des ZEO-Servers signalisiert wird, wodurch eine ZOPE-Instanz seine Schreiboperation zurückziehen und später wiederholen muss. Gibt es viele Instanzen, die auf den ZEO zugreifen, und eine Instanz kann ihre Schreiboperation nicht in 3 Versuchen erfolgreich absetzen, wird diese Operation abgebrochen und verworfen. Leider stellt aufgrund seiner alleinigen Entscheidungshoheit der einzige oder zentrale ZEO-Server einen Single Point of Failure dar, der sich mit Mitteln der ZOPE-Familie nicht beheben lässt. In der Literatur (beispielsweise in [MP02]) wird in Bezug auf diese Frage auf Softwareentwicklungen, wie „fake“ und „heartbeat“ verwiesen.

Da im Rahmen des Bildungsmarktplatzes Sachsen zwei Server mit ein und derselben Datenbasis arbeiten sollen, kommt der ZEO-Server als Speicherort für die gemeinsame ZODB zum Einsatz. Einer der beiden Rechner dient dabei jeweils zusätzlich zur lokalen ZOPE-Instanz als ZEO-Server.

Da ebenfalls in [MP02] an dieser Stelle festgestellt wurde, dass eine echte Lastverteilung nicht mit den Mitteln der ZOPE-Familie selbst erreicht wird, wurden zur Verteilung der Last folgende Vorschläge gemacht, die nach dem Aufwand zur Einrichtung aufsteigend sortiert ist:

1. passiv mittels Auswahl eines Mirrors durch den Nutzer auf einer Einstiegswebseite
2. Zufallsauswahl eines Mirrors durch einen HTTP-Redirect in einem Skript
3. Round Robin DNS, wobei hier auf Probleme des Cachings in externen DNS-Servern, die zögerliche Verbreitung von Änderungen und daraus zu bedenkende Probleme bei kurzzeitigen Ausfällen hingewiesen wird
4. Layer 4 Switching mittels entsprechender Hardware oder dem Linux Virtual Server

Die beiden letztgenannten Vorschläge sind in der Folge unter anderem noch einmal genauer und auf deren Vor- und Nachteile hin betrachtet.

Auf eine genauere Betrachtung der beiden erstgenannten Varianten wird hier verzichtet, da eine passive Auswahl von vornherein als ungünstig und nicht beeinflussbar angesehen wird, und bei der Zufallsauswahl trotz allem einer der beiden Server jeweils erst den Request bearbeiten muss, der den Redirect auslöst. Bei der passiven Auswahl neigen Nutzer nicht

vom Hauptserver zu wechseln, solange dieser keine spürbaren Beeinträchtigungen bei der Benutzung aufweist. Bei der Zufallsauswahl gäbe es theoretisch zwei Ausprägungen: Bei der ersten wird bei der ersten Anfrage ein Verweis an den Mirror geliefert und sämtlicher Restverkehr läuft über den Namen des Mirrors. Dabei sind Unausgewogenheiten im Laufe langer Sitzungen zu erwarten. Die zweite Ausprägung führt dazu, dass pro Request eine Zufallsweiterleitung stattfindet, wodurch jedoch der eine Server hauptsächlich mit der Zufallsweiterleitung beschäftigt wäre.

### **4.2.2 Dienst- und Servernamen – kein echtes Loadbalancing**

Die einfachste Form der „Lastverteilung“ ist die auf Basis einzelner Dienste und den zugehörigen üblichen Servernamen. Also dedizierte Server für einzelne Dienste bereitstellen und dann die Zugriffe darüber verteilen. Normalerweise wird ein Webserver unter dem Servernamen „www“ erwartet, ein Mailserver unter „smtp“ oder „mail“, ein Mailboxserver unter „pop“ oder „imap“ oder „mailbox“ und so weiter. Nutzt man dies nun aus und stellt einzelne Server für die einzelnen Dienste bereit, hat man bereits eine minimale Form der Lastverteilung erreicht, da prinzipiell auch ein Server alle Dienste bereitstellen könnte.

Natürlich ist die Verteilung sehr ungleichmäßig und vom Einsatz einzelner Dienste im Workflow stark abhängig. Außerdem ist die Grenze des Systems schon erreicht, wenn eine Lastverteilung über einen Dienst, beispielsweise dem HTTP-Service, funktionieren soll.

### **4.2.3 Lastverteilung mittels virtuellem Server - Switch oder LVS**

#### **Switchbasierte Lastverteilung**

Mit einem geeignet hochwertigen Switch ist es möglich, auf Basis von ISO Layer 3 oder 4 oder noch höheren den Verkehr durch den Switch und zu den angeschlossenen Geräten zu kanalisieren. Dadurch wird es ermöglicht, aufgrund von Betrachtungen des bisherigen Netzverkehrs oder Prioritäten oder einer Kombination aus beidem beziehungsweise der Anwendung bestimmter Funktionen auf Header-Informationen der einzelnen ISO-Schichten eine Verteilung des eingehenden Netztraffics auf die nachgeschalteten realen Server zu organisieren.

Dazu wird dem Switch die IP des virtuellen Servers, auf den sich die Nutzer des Dienstes verbinden, zugewiesen und dieser verteilt die Aufträge an die angeschlossenen Server und auf dem Rückweg die Antworten an die Anfragerechner. Die zur Verfügung stehenden Mechanismen sind vielfältig und exemplarisch an Hand von [Sch00] seien genannt:

1. Hash,
2. Least-Connections und
3. Round-Robin.

Beim **Hash** wird auf die Absenderadresse der Anfrage eine Funktion angewandt, die den realen Server bestimmt, an den die Anfrage weitergeleitet wird. Aufgrund der Eigenschaften von Hash-Funktionen wird dabei eine möglichst große Gleichverteilung der ankommenden Anfragen auf die realen Server erreicht. Da der Hash einer Absenderadresse immer gleich ist, ist auch das Problem von Sessiondaten damit handhabbar. Da allerdings bei Layer 3 nur die IP-Adressen als Entscheidungsgrundlage dienen, werden Netze hinter einer NAT-Firewall vollständig an einen realen Server weitervermittelt.

Beim Prinzip **Least Connections** hingegen wird auf dem Switch die Anzahl der offenen Verbindungen pro realem Server in Echtzeit gezählt. Dabei zählt eine Verbindung von der Anfragevermittlung bis zur Rückvermittlung der Antwort als offen. Der Rechner mit den geringsten aktuellen Verbindungen bekommt die nächste Anfrage vermittelt. Da dabei auch indirekt die Serverleistung der einzelnen Systeme berücksichtigt wird (ein leistungsstarker Rechner wird bis zur nächsten Anfrage mehr Verbindungen beantwortet und beendet haben und hat somit weniger offene Verbindungen als ein langsamer), hat dieser Vermittlungstypus höhere Selbstregulierungsmechanismen als der erstgenannte.

Der Dritte im Bunde macht genau das, was sein Name vermuten lässt. Beim **Round-Robin** wird die erste Anfrage an den ersten Server vermittelt. Die  $n$ -te Anfrage wird an den Server  $ServerNr = ((n \bmod (Anzahl \text{ der } Server)) + 1)$  vermittelt. Dieser Algorithmus ist einfach zu implementieren, birgt aber große Ungleichmäßigkeiten bei der Verteilung.

Laut [cis04] hat Cisco in seinem IOS die beiden Letztgenannten jeweils in Kombination mit einer Wichtung implementiert. Entsprechend der Kapazitäten der nachgeschalteten Server kann die Gleichverteilung beeinflusst werden.

### Lastverteilung durch LVS

Die Open-Source-Alternative zu einem solchen Switch mit entsprechender Server- und Lastverteilungsfunktionalität stellt das „Linux Virtual Server Project“ dar. In diesem werden die Routingeigenschaften des Linux-Kernels mit Algorithmen zur möglichst ausgewogenen Verteilung verknüpft. Die möglichen Funktionen, die bereitgestellt werden, finden sich auf [lvs04] und sind dabei:

1. NAT – Network Address Translation.
2. IP-Tunneling und,
3. Direct Routing.

Die Idee des **NAT** ist es, einen virtuellen Server einzusetzen, der für bestimmte Ports entsprechend beliebiger Schedulingalgorithmen die ankommenden Request auf diesen Port an Server weiterverteilt, die in einem von diesem Server maskierten Netz stehen.

Nachteil dieser Idee ist der Rückweg, der ebenfalls über diesen NAT-Gateway gehen muss, wobei alle Pakete anhand der NAT-Tabelle des Gateway umadressiert werden müssen. Dadurch wird der Gateway unnötig belastet und kann zum „Flaschenhals“ des Systems werden.

Die zweitgenannte Idee des **IP-Tunneling** funktioniert, indem der Lastverteiler nach der Auswahl des realen Servers ihm das empfangene Paket mittels eines IP-Tunnels zustellt. Die Antwort kann dann entsprechend des üblichen Routings des realen Servers unabhängig vom virtuellen Server geschehen.

Ganz ähnlich ist die Idee des **Direct Routing**, wobei hier statt des IP-Tunnels in den Frames der Anfrage nur die Ziel-MAC-Adresse durch die des realen Servers ersetzt wird. Dazu müssen die Interfaces der realen Server, auf denen die Anfragen eingehen, und das Interface des virtuellen Servers, auf dem die Anfragen an die realen zugestellt werden, in der gleichen ARP-Domäne liegen.

Außerdem herrschen bei beiden letztgenannten Varianten einschränkende Anforderungen an die Einstellungen der ARP-Funktionalität der realen Server auf der virtuellen IP.

#### 4.2.4 Lastverteilung mittels Round-Robin-DNS

Beim Round-Robin<sup>1</sup> (RR) im Domain Name Service nutzt man nun den Dienst aus, der den Servernamen IP-Nummern oder, ist der Servername als ein Alias gekennzeichnet, weitere Hostnamen zuordnet.

Im Falle des RR-DNS wird nun dem ersten Anfragenden nach der IP zum Servernamen die IP-Nummer des ersten Rechners ausgeliefert, dem zweiten Anfragenden die IP des zweiten Rechners und so weiter. Sind alle IPs oder Hostnamen ausgeliefert worden, wird bei der nächsten Anfrage wieder der erste Rechner genannt.

Einerseits stellt sich ein geringer administrativer Aufwand bei der Einrichtung dar, wie unter [zyt04] zu sehen ist. Demgegenüber ist zu bedenken, dass eventuelle Effekte eines Caching in entfernten DNS-Caches Auswirkungen haben können, auf die nur gering Einfluss genommen werden kann. So können ganze Netzsegmente hinter einem DNS-Cache die Lastverteilung umgehen, da dieser Cache immer wieder die eine IP ausliefern wird. Gesamtheitlich betrachtet allerdings, ist der Effekt noch ausreichend groß.

Daneben ist zu beachten, dass Sitzungsdaten, die auf den Servern eventuell gespeichert werden, dabei umgangen werden, da dieses System unabhängig von Applikationsdaten auf dem ISO Layer 4 ansetzt. Allerdings haben hier DNS-Caches (vor allem auch lokale beim Nutzer) einen positiven Einfluss, da über den Zeitraum des Caching sowieso immer die selbe IP genutzt wird.

Aufgrund des Wunsches der Hochverfügbarkeit bei diesem Projekt ist mit geeigneten Mitteln sicherzustellen, dass jederzeit auf allen zugeordneten IPs, die mittels des DNS ausgeliefert werden, ein Dienst erreichbar ist. Dazu verweise ich auf Abschnitt 5.3.1.

---

<sup>1</sup>übersetzt übrigens soviel wie „Der Plumpssack geht um“

## 4.3 Caching-Produkte

Aufgrund des Designs der Caching-Produkte und des ZOPEs ansich, ist zu bemerken, dass alle hier aufgeführten Produkte der manuellen Assoziation mit den zu cachenden ZOPE-Objekten bedürfen.

### 4.3.1 ZOPE-Standard-Cache-Manager

Standardmäßig im ZOPE vorhanden, sind zwei grundlegende Cache-Managing-Produkte, der **Accelerated HTTP Cache Manager** sowie der **RAM Cache Manager**. Aus Zeiten, da diese beiden noch als getrennte Produkte entwickelt wurden, finden sich Beschreibungen auf [std00].

Der Erstgenannte setzt dabei entsprechende HTTP-Header mit Cache-Control-Informationen, die vorher im Manager-Objekt vereinbart wurden.

Diese Informationen können einerseits clientseitig ausgenutzt werden, aber vor allem auch in externen Caches auf dem Weg zwischen Server und Client.

Besonderes Augenmerk sollte dabei auf dem Einsatz eines vorgeschalteten Cache innerhalb der eigenen Serverstruktur liegen (z.B. [apa04] / [squ04]).

Der „RAM Cache Manager“ hingegen legt einen Cache im Hauptspeicher des ZOPE-Servers an, der die mit ihm verknüpften Objekte zwischenspeichert.

Dieser wird ZOPE-seitig dazu genutzt, fertig gerenderte Objekte im Hauptspeicher bereitzuhalten und bei erneuter Anfrage sowohl das Rendering im ZOPE als auch den Datenbankquery zu vermeiden.

Aufgrund der Berichte, die sich online und in verschiedenen Literaturquellen finden, ist dies das offensichtlich einzige nutzbare und genutzte Produkt-Paket, welches weiterentwickelt und eingesetzt wird.

Die folgende Grafik zeigt, wo sich die Caches in die Serverstruktur eingliedern.

### 4.3.2 RAM-Cache-Manager with Age

Dieser Cache-Manager ist eine Erweiterung des eben erwähnten RAM-Cache-Managers aus dem StandardCacheManager-Paket.

Die Erweiterung besteht in der Einführung eines Attributs „maximum\_age“, welches die maximale Aufenthaltsdauer eines Objekts im Cache angibt. Spätestens nach Ablauf dieser Dauer wird das Objekt aus dem Cache entfernt, unabhängig von einer Aktualisierung des assoziierten ZOPE-Objekts.

Während der Tests des Produkts stellte sich heraus, dass eine aktive Entwicklung eingestellt wurde und offenbar dieses Produkt in die Standard-RAM-Cache-Entwicklung einfluss, da dieser momentan auch über die gleichen Einstelloptionen verfügt wie die „Aging“-Variante.

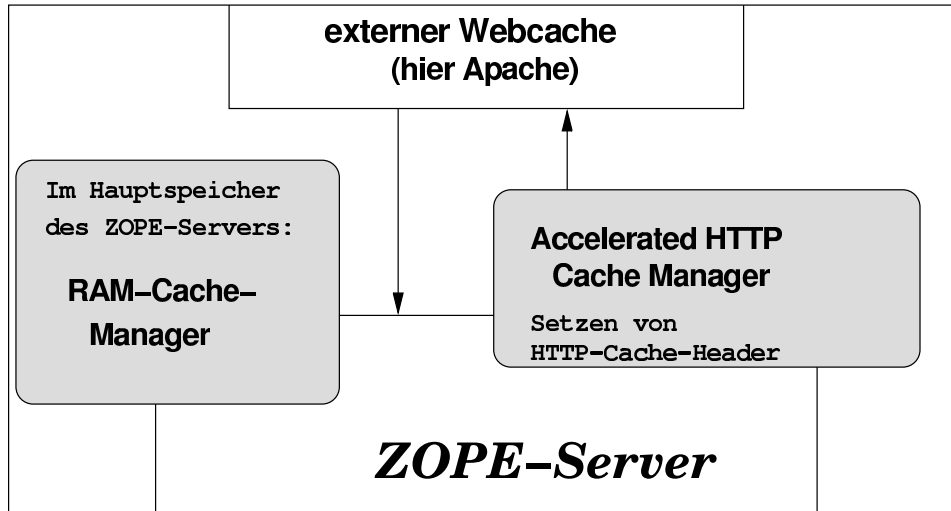


Abbildung 4.1: Sitz der Standardcache-Produkte

Hinweise zu dieser Vermutung fanden sich allerdings nicht, aber das Alter der Aussagen auf [ram01] legt dies nahe.

### 4.3.3 FS-Cache-Manager

Die Idee der Entwickler dieses Caches war es, einen externen Webserver, der dem ZOPE vorgeschaltet ist, beim Auffinden der angeforderten Objekte zu beschleunigen.

Dies geschieht dadurch, dass ein vorgeschaltener externer Webserver versucht, wenn er einen Request bekommt, das gewünschte Dokument in seiner Verzeichnisstruktur aufzufinden und auszuliefern. Sollte dies misslingen, wird der Request an den ZOPE-Webserver weitergeleitet.

Der „FS-Cache-Manager“ greift in dieses Verfahren nun dadurch ein, dass eine Kopie des gerenderten und als Dokument ausgelieferten ZOPE-Objekts im Filesystem des Webservers abgelegt wird, wenn ein Request durch den ZOPE-Server bearbeitet wurde. Beim nächsten Zugriff auf diese URL kann nun der externe Webserver das Dokument direkt aus dem Filesystem ausliefern.

Die Aktualität der gecachten Objekte wird durch Löschen der Kopien im Filesystem gesichert, sobald das assoziierte ZOPE-Objekt geändert wird.

So interessant die Idee ist, wurde sie offenbar nie zu einer nutzbaren Implementierung vollendet. Die auf [fsc02] zum Download bereitgestellten Versionen sind fehlerhaft und nicht einsetzbar. Die Entwickler verweisen auf den Einsatz des Standard-Cache-Pakets.

Einzig im Kontentor-CMS (siehe dazu [kon03b]) wurde die Idee laut [Ric03] aufgegriffen, aber wahrscheinlich auch nochmals neu implementiert. Hinweise finden sich in [kon03a],

jedoch keinerlei Ausführungen über die Implementierung. Auch in der Installation selbst finden sich keine offensichtlichen Optionen.

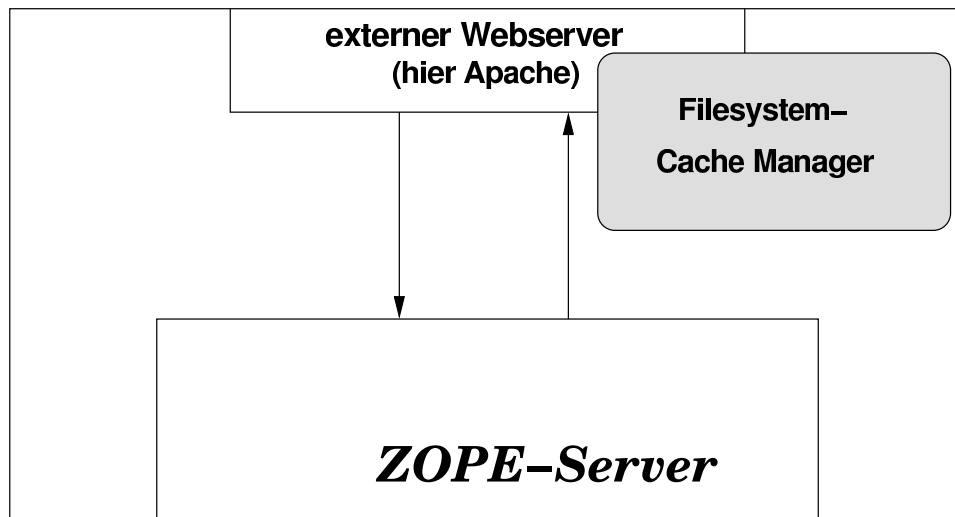


Abbildung 4.2: Sitz eines Filesystem-Caches

#### 4.3.4 ZOPE-Proxy-Cache-Manager

Das auf [per01] beschriebene Produkt verspricht ein schnelles Caching, da es voll im externen Cache durchgeführt wird. Daher entfällt wieder der Aufwand für aufwendige Datenbank-Anfragen an die ZOPE-DB und die Notwendigkeit des neuen Renderings.

Der externe Cache muss in diesem Falle ein Apache-Webserver sein, da dieser Cache auf Basis des `mod_perl`<sup>2</sup> implementiert ist.

Laut Angaben der Entwickler werden private Elemente von Dokumenten beim Caching beachtet. Genannt sind hier Cookies, Parameter und Authentisierungsinformationen in und zu Dokumenten.

Der Aufwand für die Installation ist groß und die Komponenten, deren Funktion sichergestellt werden muss, sind deutlich zahlreicher als bei anderen Cache-Produkten.

Eine Installation entsprechend der dem Produkt beiliegenden Anleitungen war erfolglos und es wurden keine Dokumente im Dateisystem abgelegt. Dabei wurden Tests mit verschiedenen Objektarten durchgeführt, die alle erfolglos blieben.

---

<sup>2</sup>Siehe dazu: <http://perl.apache.org>

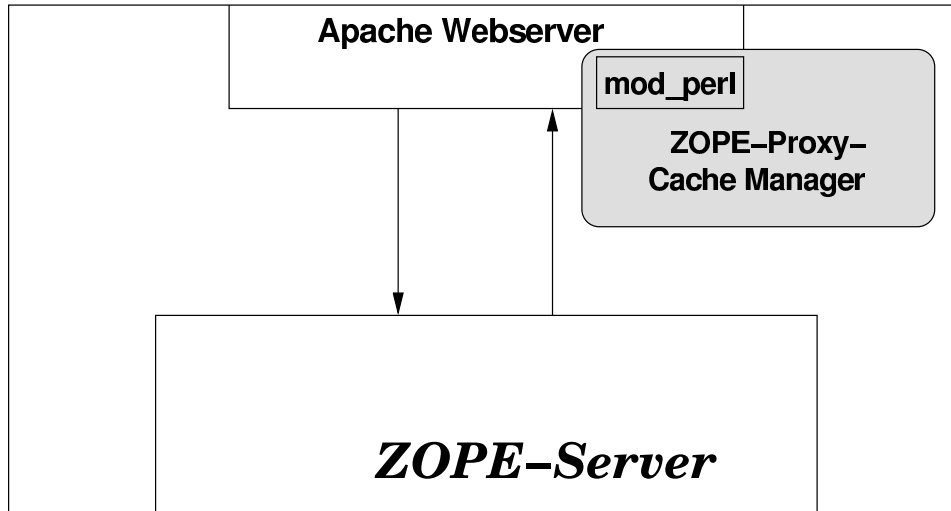


Abbildung 4.3: Anordnung mit dem Zope-Proxy-Cache-Manager

### 4.3.5 Cache Controlled ZSQL Methods

Eine interessante Idee wurde auf [ccz04] vorgestellt.

Der dort beschriebene Cache legt einen Zwischenspeicher für die Ergebnisse von SQL-Anfragen innerhalb des Zope-Servers an. Die Anfragen werden samt Parametern eingelagert und entsprechend derer auch erneut ausgeliefert.

Das Ziel ist es, komplexe und häufige Anfragen an die Datenbank zu minimieren.

Eine Aktualisierung des Caches erfolgt nicht automatisch mit der Aktualisierung der Datenbank oder eines anderen Objektes. Dafür ist ein expliziter Funktionsaufruf notwendig.

Daher empfehlen die Entwickler, diesen Funktionsaufruf beispielsweise im Rahmen eines Triggers in der Datenbank ereignisgesteuert ausführen zu lassen.

Außerdem sind zwei Funktionsrufe für die Cachemanipulation implementiert, wobei der Ruf `your_ccsql_method.flushCacheEntry()` auch die Parameter der gecachten SQL-Anfragen entgegennimmt und nur solche Einträge löscht, die dazu passen.

Dieser Funktionsruf wird allerdings nur in der lokalen Zope-Installation signalisiert, während der Funktionsruf `your_ccsql_method.flushCache()` zwar keine Argumente entgegennimmt und somit immer den vollen Cache des assoziierten Objekts entleert, aber bei einer ZEO-Installation über die lokalen Servergrenzen hinweg in die komplette ZEO-Server-Struktur signalisiert wird. Daher käme im Umfeld des Themas dieser Arbeit nur die zweite Methode in Betracht.

Da im Rahmen des Projektes auf die Nutzung der ZSQL-Methoden verzichtet wurde und der Zugriff auf die PostgreSQL-Datenbank aus eigenen Skripten erfolgt, ist ein Einsatz dieses Produkts derzeit nicht möglich. Daher habe ich keine nähere Betrachtung des Produkts

vorgenommen.

In der Folge der Tests hat sich gezeigt, dass zwischen den Standard-ZSQL-Methoden und diesem Produkt keine Unterschiede messbar sind und offenbar die beiden Produktfamilien vereinigt wurden. Gleiche Einstellmöglichkeiten in den Produkten legen den Gedanken nahe, dass die in der Entwicklung eingestellten CCSQL-Methoden mit in die Entwicklung der ZSQL-Methoden einfließen.

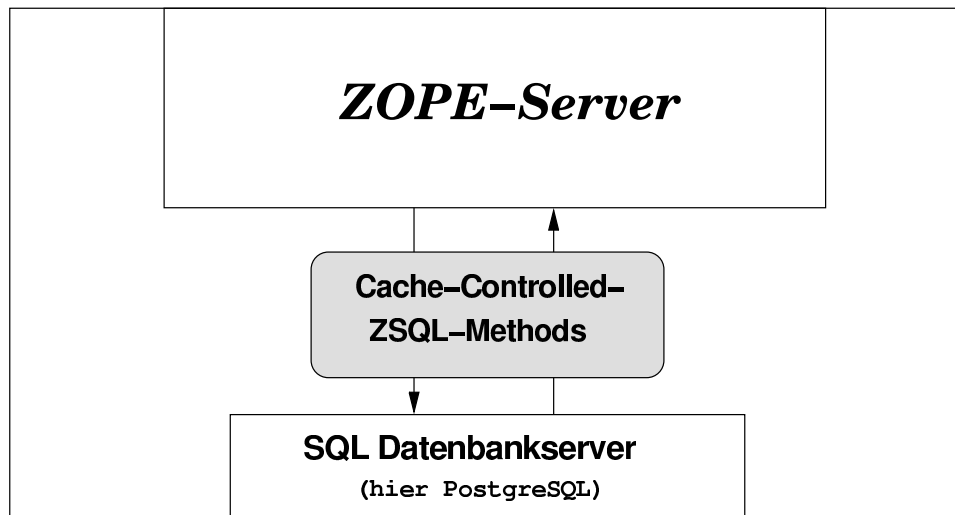


Abbildung 4.4: Cache Controlled ZSQL Methods

#### 4.3.6 Die Produkte im Vergleich

Wie in der Tabelle 4.1 zu sehen ist, bleibt einzig der Vergleich der beiden Standardcaching-Produkte als sinnvolle Auswahl. Die ZSQL-Methoden haben kaum messbare Unterschiede zu den Cache Controlled ZSQL-Methoden, da wahrscheinlich kein Unterschied mehr zwischen den Produkten besteht.

Fraglich bleibt weiterhin, wie der Wunsch, rollenspezifisch zu cachern, realisiert werden kann. Sinnvoll wäre ja ein Caching für CMS-Elemente von allen Rollen außer dem Manager und Reviewer. Der HTTP-Cache kann allerdings nur zwischen anonymen und authentifizierten Requests unterscheiden. Damit wäre ein Caching für Interessenten solange einfach realisierbar, wie diese nicht angemeldet auf das Angebot zugreifen. Ähnlich verhält es sich mit dem RAM-Cache, dessen Steuerung aufgrund von ZOPE-Variablen stattfindet.

Die Messreihe, aus der die Tabelle 4.1 entstand, verlief mit Hilfe des Apache Benchmark Tools „ab“. Die Spalten von links nach rechts bezeichnen das Objekt, welches abgerufen

Objekt	Produkt	Median		Requests / Sekunde		Median-Speedup
		ohne Produkt	mit Produkt	ohne Produkt	mit Produkt	
Pagetemplate	RAM-Cache	90	80	275.27	306.46	1.125
Pagetemplate	HTTP-Cache	71	4	138.66	2019.61	17.75
Pagetemplate	RAM-Cache with Age	90	81	275.27	304.28	1.111
Pagetemplate	FS-Cache	90	115	275.27	215.29	0.783
Template mit ZSQL-Methode	RAM-Cache / ZSQL	357	60	27.77	164.23	5.950
Template mit ZSQL-Methode	HTTP-Cache / ZSQL	513	5	19.36	1817.34	102.6
Template mit CCZSQL-Methode	RAM-Cache / CCZSQL	358	58	27.73	168.61	6.172
Template mit CCZSQL-Methode	HTTP-Cache / CCZSQL	519	5	19.12	1844.18	103.8
Pagetemplate (über WWW)	RAM-Cache	154	141	132.35	126.60	1.092
Pagetemplate (über WWW)	HTTP-Cache	154	137	132.35	143.11	1.111

Tabelle 4.1: Cacheproduktevergleich

wurde, dann die Produkte, welche zum Cachen zum Einsatz kamen. Der Median ist die mittlere Antwortzeit auf einen Request und die Requests pro Sekunde beschreiben die Aufnahmekapazität des Servers. Der Speedup ist das Verhältnis von Antwortzeit ohne und mit Produkt, wie in 2.2 erläutert.

Da die Messungen der verschiedenen Produkte auf mehreren Rechnern entstanden, sind verschiedene Rahmenbedingungen eingesetzt worden. Die Messungen der einzelnen Produkte, vor und nach deren Einsatz, fanden jedoch immer unter den selben Bedingungen statt, so dass die absoluten Zeiten zwischen den Messungen nicht vergleichbar sind, aber die relativen Speedups.

Zur Tabelle sei außerdem erklärt, dass die letzten beiden Zeilen von einem entfernten Rechner gemessen wurden, während die restlichen Zeilen eine Messung direkt vor dem Server darstellen. Sinn der Messung war, den Einfluss von Webproxys auf dem Weg zum Nutzer zu betrachten.

Aufgrund der gemachten Beobachtungen kann ich mich nur [McK04] anschließen und eine einfache Einteilung für das Caching vorschlagen. Dabei sollten komplexe Skripte, die auf die Datenbank zugreifen, mit dem RAM-Cache assoziiert werden, damit eine Entlastung beim Rendering und der Datenbankabfrage genutzt wird. Für statische Elemente, wie Bilder und andere Skin-Elemente, sollte eine Verknüpfung mit dem HTTP-Cache vorgesehen werden, um eine Beschleunigung des Downloads beim Endkunden zu erreichen. Eine weitere sinnvolle Idee folgt in 4.3.8.

### 4.3.7 Darüber hinausgehende Entwicklungen von Caching-Produkten

Wie aus den bereits vorhandenen Caching-Produkten bzw. Produkt-Ideen ersichtlich ist, gibt es eine Vielzahl von Versuchen, an geeigneten Eingriffspunkten anzusetzen. Gleichzeitig ließ sich aufgrund der Entwicklungsstadien und -umsetzungen erkennen, dass viele der Ideen mit der Zeit verworfen wurden oder aufgrund fehlender Einsatzszenarien die Implementierung aufgegeben wurde.

Viele der begonnenen Ideen sind jedoch für stark belastete Systeme interessant. So ist die Idee des Cachings im Filesystem mehrfach aufgekommen und ansatzweise (in [kon03b] wohl auch intensiver) umgesetzt worden, da eine Auslieferung einer statischen Webseite durch einen Webserver schnell und kostengünstig geschehen kann.

Auch interessant ist die Idee der Zwischenspeicherung von Datenbankabfragen. Doch wäre angesichts der gemachten Ergebnisse von Stichprobenmessungen im Rahmen dieser Arbeit zu überprüfen, ob die Abarbeitung der Anfragen überhaupt eine derartige Pufferung sinnvoll macht. Doch gerade hinsichtlich von Anfragen mit komplexer Tabellenstruktur wäre eine Betrachtung dieses Umstands interessant.

### 4.3.8 Contentbasiertes Caching im Apache

Aufgrund der recht beschränkten Einsatzmöglichkeiten der ZOPE-Produkte auf dem Feld des Cachings ist eine weitere Überlegung interessant. Dabei wird im Reverse-Proxy vor dem ZOPE-Server nicht ausschließlich auf Grundlage der gelieferten Cache-Control-Header zwischengespeichert, sondern auch aufgrund der Informationen über den Content-Type des gelieferten Dokuments, wie es beispielhaft auf [apc04] beschrieben ist. Dabei sind Einstellungen sinnvoll, die vorsehen, Grafiken und statische Elemente des Portals, wie Stylesheets, lange zu lagern und aus dem Cache zu liefern, während die restlichen Elemente anhand der Cache-Control-Einstellungen des ZOPE-Servers behandelt werden.

Vorteil dieses Cachings ist die Kombination von diesem Content-type-basierten Caching mit den sonstigen Cachingmöglichkeiten des ZOPEs. Nachteil ist die fehlende Beeinflussung vom ZOPE aus und der Aspekt, dass gegenüber dem Accelerated HTTP Cache Manager von dieser Strategie keine Webcaches auf dem Weg zum Endnutzer profitieren.

## 4.4 Wie kann eine hohe Verfügbarkeit erreicht werden?

Um Aussagen über die notwendigen Eingriffe machen zu können, ist es erst einmal notwendig herauszustellen, welche Dienste nötig sind und welche Abhängigkeiten zwischen den Diensten bestehen oder welche Grundlagen für das Funktionieren eines Dienstes gegeben sein müssen.

### 4.4.1 Basisüberlegungen

Hier folgt eine kurze Vorstellung der Ideenansätze, die eine hohe Verfügbarkeit der einzelnen Dienste erreichen können.

Wird der Ausfall einer einzelnen ZOPE-Instanz festgestellt, kann versucht werden, diese automatisch neuzustarten. Scheitert dies, müssen mittels geeigneter Maßnahmen die Zugriffe, die bisher an diese Instanz vermittelt wurden, an den anderen Server umgeleitet werden. Dies gilt äquivalent für den Ausfall eines der Apache-Webserver.

Fällt der ZEO-Server aus und lässt sich dieser nicht schnellstmöglich neustarten, muss auf eine andere Instanz eines ZEO-Servers auf dem zweiten Server umgeschaltet werden, wobei dieser gleiche Inhalte anbieten soll.

Dazu muss regelmäßig, optimalerweise ständig, ein Abgleich der Inhalte der vom ZEO-Server verwalteten ZODB stattfinden. Im Falle des Ausfalls muss der Zugriff der ZEO-Klienten, also den vorgeschalteten ZOPE-Instanzen, auf den nun aktuellen ZEO-Server umgeleitet werden.

Ähnlich verhält es sich bei Ausfall des Datenbankservers, da dieser ebenfalls zentral Inhalte vorhält, die für den Betrieb unablässig sind.

Auch hier muss bei Scheitern eines Neustarts auf den zweiten Server verlagert werden und eine Umleitung der Zugriffe erfolgen. Replikation und Rekonfiguration seien als Stichworte genannt.

Kommt es zum Ausfall eines Netzwerkinterfaces müssen sämtliche Funktionen des ausgefallenen Servers auf den zweiten Server migriert und die Zugriffe ebenso umgeleitet werden. Einfach gestaltet es sich, wenn der Server nur eine ZOPE-Instanz vorhielt. War er jedoch DB- und ZEO-Server, müssen alle Funktionen mit minimalen Datenverlusten dem zweiten Server übertragen werden.

### 4.4.2 Zugriffsumleitung

Um die Zugriffe auf einen Server umzuleiten, bestehen prinzipiell mehrere Möglichkeiten.

1. Mittels Anpassung des DNS-Records.
2. Wäre der Einsatz eines Lastverteilers oder virtuellen Servers denkbar.
3. Durch die Übernahme der IP des ausgefallenen Rechners.

Bei der Anpassung des DNS-Records wird auf den Umstand zurückgegriffen, dass in den meisten Fällen mittels der Hostnamen und nicht mittels der IP adressiert wird. Dabei wird dem Hostnamen des ausgefallenen Rechners die IP des Servers zugeordnet, der den Dienst übernimmt.

Dabei ist zu beachten, dass die weltweite Verteilung und das Caching von DNS-Einträgen auf anderen Servern und in lokalen DNS-Caches zu großen Verzögerungen bis zur vollständigen Bekanntmachung der Umstellung führen kann – Zeiten im Bereich mehrerer Stunden sind üblich. Während dieser Zeit werden Anfragen immer noch an die IP des ausgefallenen Rechners vermittelt und die Dienste scheinen für den Nutzer nicht erreichbar. Im schlimmsten Falle ist sogar vorzustellen, dass einzelne Nutzer die Bekanntmachung erst erfahren, nachdem der ausgefallene Rechner bereits wieder instand gesetzt wurde. Es wäre auch denkbar, dass in der Folge der Server, der den Dienst übernommen hatte, noch Stunden nach der Reparatur des ursprünglichen Zielrechners mit Anfragen beschäftigt wird, die für den anderen Rechner gedacht sind – und das zusätzlich zu den ihm übertragenen Aufgaben.

Neben dieser recht einfach zu implementierenden, aber mit oben genannten unschönen Nebeneffekten behafteten Methode wäre ein Szenario vorstellbar, in dem ein den Server vorgeschaltener virtueller Server die Verteilung auf die dienstbringenden Rechner koordiniert. Dabei sind die unterschiedlichen Möglichkeiten denkbar, die beispielsweise das Linux Virtual Server Project oder die verschiedensten Hardware-Router (beispielsweise von Cisco)

bereitstellen.

So könnte der virtuelle Server die Erreichbarkeit der Dienste auf den eigentlichen Servern testen und demnach die Verteilung vornehmen. Da hierbei jeglicher Verkehr über die IP des virtuellen Servers abgewickelt wird, wäre der Ausfall für die Nutzer transparent. Auch ließe sich dabei der Einsatz von lastbasierten Verteilungsmechanismen vorstellen.

Da aber der virtuelle Server hierbei einen Single Point of Failure darstellt, sollte die Implementierung im Produktivbetrieb robust sein – eventuell robuster, als ein Linux Virtual Server ist – und die Möglichkeit dieser Ausfallstelle bedacht werden.

Die dritte Möglichkeit, die eine Umleitung der Anfragen, die eigentlich an einen ausgefallenen Server gerichtet sind, ermöglicht, ist die Übernahme der IP durch den Ersatz-Rechner. Dabei erhält der vorübergehende Dienstleister die IP, an die ursprünglich der Traffic gerichtet ist.

Diese Umschaltung kann aktiv geschehen, indem der Ersatz-Rechner die IP zusätzlich zu seiner oder seinen bisherigen auf ein Interface schaltet oder auch indem er mittels arp-spoofing nur die ARP-Anfragen an die IP beantwortet, dann allerdings auf IP-Ebene mit seiner eigenen arbeitet. Da die einzelnen Schichten allerdings verschiedene Sicherungsmaßnahmen zur Entlastung des eigenen Rechners und vor Angriffen oder Fehlkonfigurationen anderer Rechner enthalten, würden Pakete mit fehlerhafter Adressierung verworfen. Somit müsste man diese Sicherungsmaßnahmen ausschalten und umgehen, sowie in der Folge für deren Funktionalität eigene Vorkehrungen treffen, weshalb ein zusätzliches Beschalten eines Interfaces mit der zu übernehmenden IP einfacher, sicherer und stabiler ist.

### 4.4.3 Rekonfiguration

Geht man von der einfachsten Einstellung aus und nimmt an, die beiden Server des Bildungsmarktplatzes Sachsen hätten die IP *A* und *B*, wobei beispielsweise der Server *A* zusätzlich den ZEO-Server und Datenbankserver beheimatete, wären die naiven Konfigurationen in den ZOPE-Instanzen so, dass alle Instanzen auf den ZSS mit der IP *A* zugreifen. Käme es zum Ausfall auf Server *A*, müsste eine ZEO-Instanz auf dem Server *B* gestartet werden und beide ZOPE-Instanzen so umkonfiguriert werden, dass sie auf den ZEO auf Server *B* zugreifen. Dazu wäre ein Stoppen, Umschreiben der Konfigurationsdatei und ein Neustart des Dienstes nötig.

Gerade aber das automatische Umschreiben von Dateien ist heikel, da diese strengen syntaktischen Vorgaben genügen müssen.

Allerdings bietet dieses Szenario eine hohe Sicherheit beim Failback, da jeder Server nur seine IP nutzt und keine Probleme mit eventuell bereits genutzten IPs entstehen können.

Nutzt man aber die Zugriffsumleitungen auf Basis der IP-Übernahme oder des vgeschalteten virtuellen Servers, kann man sich diesen Punkt ersparen und trägt als IP oder Hostnamen der wichtigen Dienste einen Wert ein, der immer dem aktiven Master zugesichert

ist (zum Beispiel die IP, die im Ausfall übernommen wird, oder eben die des virtuellen Servers, der sich um Weiterverteilung anhand seiner Einstellungen kümmert).

#### 4.4.4 Replikation

Bleibt noch die Frage: Wie kann man dafür Sorge tragen, dass selbst bei Ausfall eines einzelnen Servers, keine oder nur wenige Daten verloren gehen?

Eine Möglichkeit ist es, wie es mehrere kommerzielle Produkte tun, auf die Notwendigkeit eines Speichermediums zu bestehen, welches für alle beteiligten Server erreichbar ist - SCSI-RAID, SAN, NAS seien hier als Beispiele genannt.

Die andere Möglichkeit besteht darin, mittels geeigneter Mechanismen die Inhalte über mehrere Rechner redundant zu halten, um bei einem Ausfall des Servers, der die Inhalte primär bereitstellt, auf die Kopie eines anderen Rechner zurückgreifen zu können.

Das Konzept des ZOPE-Servers hält nur Sitzungsdaten im Hauptspeicher, alle persistenten Daten der Anwendung werden in der ZODB abgelegt. Im Projekt Bildungsmarktplatz Sachsen werden viele weitere Daten in der zentralen PostgreSQL-Datenbank abgelegt. Somit sind die ZODB und die Datenbank des PSQl-Servers, die Elemente, die so aufbewahrt werden müssen, dass ein Verlust eines Speichermediums nicht zum Datenverlust im großen Umfang führen und nicht erst ein Backup-Dump zurückgestellt werden muss.

#### Datenreplikation am PostgreSQL

Beim Postgres gibt es hierbei grundsätzlich zwei Ansätze, die Replikation mit Hilfe von Datenbankmitteln durchzuführen oder auf Werkzeuge zur Dopplung der Speicherplätze im Dateisystem zurückzugreifen.

Die offensichtlichsten Mittel, die die Datenbank selbst zur Verfügung stellt, sind Trigger, wobei im Falle einer Datenmanipulation diese dem Duplikatserver mitgeteilt und auf dessen Datenbestand ebenfalls angewandt werden muss.

Auf Basis des Filesystems bieten sich die verschiedensten Mittel, wie „rcp“, „scp“, das Einbinden von Netzlaufwerken und dann eine das Nutzen von „cp“. Natürlich bietet sich das als „scp-Framework“ nutzbare „rsync“ besonders an.

Allerdings bleibt beim Kopieren auf Dateiebene immer das Problem von geöffneten Files, die beispielsweise der Datenbankserverprozess während seiner Laufzeit nicht freigibt und die daher nicht kopiert werden können.

Produkte im PostgreSQL-Umfeld, die für die Replikation interessant klangen und daher näher betrachtet wurden, sind

1. pgpool
2. SQL-Relay
3. Slony-1

Meist versprechendes Produkt war pgpool, welches sowohl eine Lastverteilung als auch Replikation über PostgreSQL-Datenbanken versprach.

Das Selbstverständnis des Produkts findet sich auf der Webseite wie folgt:

pgpool is a connection pool server for PostgreSQL. pgpool runs between PostgreSQL's clients(front ends) and servers(back ends). A PostgreSQL client can connect to pgpool as if it were a standard PostgreSQL server.

pgpool caches the connection to PostgreSQL server to reduce the overhead to establish the connection to it.

Also, pgpool could use two PostgreSQL servers for fail over. If the first server goes down, pgpool will automatically switch to the secondary server.

Es war einfach zu installieren und konfigurieren. Es kann zwei Server verwalten, wobei ich mir eine Kaskadierung vorstellen kann, da sich das System wie ein transparenter Proxy verhält. Die angewandte Technik besteht in der Duplikation der Requests auf alle Backends. Dabei können zwei Modi gewählt werden, der strict-mode, bei dem der Request erst auf den Slave dupliziert wird, wenn die Operation auf dem Master erfolgreich war, und der echt parallele, wo die Duplikation sofort auf beide Server stattfindet. Weiterhin erkannte das System, wenn der Master nicht mehr erreichbar war und führte die Operationen dann ausschließlich auf dem Slave aus.

Probleme, die das System mit sich bringt, sind unter anderem, dass die versprochene Lastverteilung ausschließlich auf Leseoperationen ausgeführt werden kann und um dies zu sichern, funktioniert diese nur bei SELECT-Statements. Bereits das BEGIN einer Transaktion, ob lesend oder schreibend, unterbindet das Lastverteilungsverfahren, und die Operationen werden nacheinander auf Master und Slave durchgeführt. Weiterer und wesentlicherer Punkt ist der Fakt, dass zwar ein Ausfall des Masters erkannt und geloggt wird, allerdings keine Auszeichnung des Slaves als Master erfolgt, was bei einem Neustart des Masters einen Rücksprung auf dessen Datenbestand bedeutet. Es erfolgt keine Plausibilitäts- und auch keine echte Konsistenzprüfung der Daten.

Entscheidendes Ausschlusskriterium war der Umstand, dass eine kurzzeitige Unterbrechung der Erreichbarkeit des Slaves vom Master aus zu Dateninkonsistenzen führt, die nicht erkannt und behoben werden.

SQL-Relay ist ein Produkt, welches ebenfalls wie ein Proxy zwischen Anwendung und Datenbank liegt und speziell für die Performanzsteigerung von Webanwendungen programmiert wurde.

Eine grobe Beschreibung des Produkts findet sich auf [sql04]:

What is SQL Relay?

SQL Relay is a persistent database connection pooling, proxying and load balancing system for Unix and Linux supporting ODBC, Oracle, MySQL, mSQL, PostgreSQL, Sybase, MS SQL Server, IBM DB2, Interbase, Lago and SQLite with APIs for C, C++, Perl, Perl-DBD, Python, Python-DB, Zope, PHP, Ruby, Ruby-DBD and Java, command line clients, a GUI configuration tool and extensive documentation. The APIs support advanced database operations such as bind variables, multi-row fetches, client side result set caching and suspended transactions. It is ideal for speeding up database-driven web-based applications, accessing databases from unsupported platforms, migrating between databases, distributing access to replicated databases and throttling database access.

In [srf04], der FAQ des Produkts, findet sich eine Begründung, weshalb ein Einsatz mit ZOPE interessant sein könnte:

Why should I use SQL Relay with Zope?

The same efficiency arguments that can be made against Apache::DBI and PHP's persistent connections cannot be made against Zope. Zope maintains a hackable (some say "configurable") number of persistent database connections in its cache and shares them among its threads. The number of database connections and threads are independent. There is always the possibility that one or all of the database connections will get pushed out of the cache and have to be started back up later, but in practice, this is highly unlikely and happens very infrequently.

If you have such a large farm of Zope machines that the number of persistent database connections is straining the database server's resources, SQL Relay can provide a middle tier to reduce the number of persistent connections.

SQL Relay adds immediate support for load distribution over a group of clustered or replicated databases to Zope.

SQL Relay can provide a means for connecting to databases for which there is no Zope adapter.

Dieses Produkt ist aber eben ausschließlich auf Lastverteilung spezialisiert und unterstützt keinerlei Form der Datenreplikation, die wiederum anderen Produkten überlassen wird, wie sich in einer anderen Aussage der FAQ findet. SQL-Relay unterstützt laut FAQ zwar schreibende Funktionalität auf einen „Master“ zu beschränken, wenn dies die Replikation benötigt. Erschwerend ist nur, dass keinerlei Gewähr für die Synchronisation gegeben werden kann und auf externe Werkzeuge verwiesen wird. Dabei ist bedenkenswert, dass sich die Lastver-

teilung eventuell erschwerend auf die Replikation auswirkt.

*Jan Wieck* hat Mitte dieses Jahres das erste Release seines Werkzeugs „Slony-1“ freigegeben, welches ein Framework zur Replikation speziell von PostgreSQL-Datenbanken bereitstellt.

Slony-1 arbeitet in Form eines Daemons, der auf Events der Datenbank wartet und daraufhin Skripte ausführt. Dabei wird im Slony-1 ein Cluster über mehrere Datenbanken, üblicherweise auf verschiedenen Servern, definiert und der Zugriff auf diese geregelt.

Ist dies geschehen, werden Datenmanipulationen (bisher keine Schemata-Manipulationen) nach Definitionen des Skriptes auf alle Knoten des Servers dupliziert.

Slony-1 baut zur Verwaltung der Replikation eine Extra-Datenbank auf, in der die Manipulationen abgelegt werden, die noch auf die Slaves übertragen werden müssen. Das Produkt arbeitet strikt nach dem Master-Slave-Prinzip und bietet keine Möglichkeit des gegenseitigen Abgleichs, also einer Replikation von allen beteiligten Servern auf alle restlichen im Cluster.

Trotz dieser kleineren Mankos stellt diese Software die günstigste Lösung für die Replikation der PostgreSQL-Datenbank dar. Dies liegt in den Eigenschaften begründet, wie etwa einer Unterstützung mehrerer Slaves. Ein weiterer wesentlicher Punkt ist, dass ein kurzzeitiger Ausfall der Erreichbarkeit des Slaves durch den Master nicht zu einem Abbruch der Replikation oder einer Dateninkosistenz führt. Vielmehr wird aufgrund der Verwaltungsdatenbank, die das Slony-1 auf dem Master einrichtet, für jeden Slave vermerkt, welche Daten noch zu replizieren sind. Darüber hinaus bietet das Werkzeug eine Funktion, die sehr sinnvoll ist und seine Bewertung weiter steigen lässt. Slony-1 unterstützt eine Funktion zur dynamischen Vergabe der Rollen Master und Slave. So kann im Falle eines Wartungsfensters für den Master einem Slave (in diesem Falle gibt es ja nur einen) die Rolle des Masters übergeben, danach der nun neue Slave und alte Master bearbeitet werden. Beim Failback wird der alte Master als Slave eingebunden, ein Datenupdate mittels Slony-1-Replikation angestoßen und danach kann eine Übergabe der Rolle wieder als Master erfolgen.

### **Replikation der ZODB**

Im Falle der ZODB ist der Ansatz ähnlich, wobei es keine datenbankeigenen Mechanismen gibt. Somit bleibt die Replikation nur mittels Kopie auf Dateisystemebene, auch wieder mit oben genannten Werkzeugen wie rsync oder Ähnliche, oder als Alternative der Einsatz zusätzlicher Produkte.

Die Zahl dieser Produkte ist recht gering. Ein kommerzielles Produkt, welches hohe Zuverlässigkeit, automatische Replikation und uneingeschränkte Nutzbarkeit für alle Objekte der ZODB verspricht, ist „ZOPE-Replication-Service“. Dieses hat allerdings einen sehr hohen Preis – schon ca. 10000\$ für die erste CPU, auf der eine ZODB gemanaged werden soll.

Ein freies Produkt, welches Daten über mehrere ZOPE-Instanzen und deren ZODB-Speicher abgleicht, ist „ZSyncer“. Dieser wird als Produkt installiert und kann dann in die Objekt-Hierarchie im ZOPE eingebunden werden. Danach wird über die Management-Webseite der Zsyncer mit den notwendigen Daten ausgestattet, welche den Server, mit dem verglichen werden soll, und den Nutzer, mit dem darauf zugegriffen wird, benennen sowie die abzugleichenden Objekttypen festlegt. Außerdem kann eingestellt werden, ob oder ob nicht rekursiv durch den folgenden Objektbaum gegangen wird. Ist dies alles festgelegt, kann mittels eines URLs in der Objekt-Hierarchie ein Vergleich des jeweils aktuellen Objektstandes gemacht werden und gegebenenfalls ein Webformular den Abgleich anstoßen. Dieses Webformular kann nun wiederum automatisch unter Zuhilfenahme von Skripten abgerufen und ausgewertet werden, so dass sich ein automatisierter Abgleich beispielsweise unter Verknüpfung mit Werkzeugen wie wget und der Shell periodisch erreichen lässt.

Darüber hinaus bleibt noch ein Python-Skript, welches der ZOPE-Distribution beiliegt und sich „repozo.py“ nennt. Im beigelegten README-File wird es nur kurz mit der Beschreibung „incremental backup utility for FileStorage“ gewürdigt. Dieses Skript in den Python-Baum des ZOPE-Servers kopiert, kann es eine Kopie einer „Data.fs“-Datei erstellen und auf Wunsch auch mittels gzip komprimieren. Mit dem selben Skript ist eine Wiederherstellung von derart erstellten Backups zu einem funktionalen Data.fs-File. Durch den Zugriff des Skripts direkt mittels der ZODB.FileStorage-Bibliothek kann es beim Einsatz nicht zu Problemen mit Locks auf Filesystem-Ebene kommen. Außerdem ermöglicht dieses Tool ein inkrementelles Backup, was die Zeiten für das Backup deutlich verkürzen dürfte. Natürlich ist es allein nicht geeignet, über die beiden Maschinen hinweg zu arbeiten. Doch wäre ein Einsatz beispielsweise mittels Netzlaufwerken, auf die beide Maschinen Zugriff haben denkbar. Dabei wäre ein Einsatz eines Cronjobs auf dem Master denkbar, welcher regelmäßig einen Abzug auf ein Netzlaufwerk schreibt und entweder ein Cronjob auf dem Slave, der den Abzug von dort liest und einpflegt oder ein Skript, welches eine Art Daemon darstellt und wartet, bis ein Abzug abgelegt und freigegeben wurde.

## 4.5 Überwachungsmechanismen

Nachdem nun geklärt ist, welche Möglichkeiten gegeben sind, einen Komponentenausfall zu kompensieren, muss ich eigentlich eingestehen, den vielleicht zweiten Schritt vorm ersten getan zu haben. Denn eigentlich kommt vor dem Servicetransfer die Feststellung der Notwendigkeit hierzu. Diese wiederum kann nur durch entsprechendes Monitoring der Server und Dienste stattfinden. Daher will ich nun kurz darauf eingehen, welche Möglichkeiten zur Überwachung bestehen.

Schaut man in der Literatur rundum ZOPE zu Beiträgen zum Thema Hochverfügbarkeit wird meist primär auf den Einsatz von ZEO verwiesen, um mehrere ZOPE-Instanzen mit

gleichen Inhalten bereitstellen und damit diesen ersten Single Point of Failure beseitigen zu können. Sollte Bedarf an höherer Ausfallsicherheit und Redundanz des ZEO-Servers bestehen, zählt beispielsweise [MP02] externe Produkte wie

- „fake“ oder
- „heartbeat“

auf. Der Zweitgenannte scheint der verbreitetste Vertreter der Kategorie zu sein, die allerdings mit den genannten nicht ausgeschöpft ist. Weitere Produkte, die ähnliche Funktionalität bereitstellen, sind für die Linux/Unix-Systemwelt beispielsweise „mon“ oder auch „fping“ mit eigenen Skripten. Im neuesten Serverbetriebssystem aus Redmont, Windows 2003-Server, finden sich zum Beispiel systemeigene Erweiterungen, um die gewünschte Funktionalität zu erreichen. Hier seien der „Netzlastenausgleich“ und die „Microsoft-Cluster-Komponente“ als Stichpunkte genannt.

Darüber hinaus besteht jederzeit die Möglichkeit, die Überwachung und in deren Folge die Umschaltung mit Hilfe eigener Programme oder Skripte ausführen zu lassen. Gerade im Bereich Linux/Unix stellt die Shell ein mächtiges Werkzeug dar und mit ihr ließe sich in Skripten das Gewünschte, zum Beispiel durch Einsatz und Kombination von Programmen wie ping, ps und ifconfig, erreichen.

#### **4.5.1 heartbeat**

„heartbeat“ ist das zentrale Element der Linux-Hochverfügbarkeitsplattform des „Linux HA Projects“. Dieses auf [lin04] gehostete Softwarepaket umfasst funktional die Überwachung eines Rechnerclusters sowie ausgewählter Prozesse auf dem Server, der aktuell aktiv ist und den Service bedient. Darüber hinaus bietet „heartbeat“ selbst bereits die Funktionen zur Umschaltung und zum Diensttransfer bzw. Starten und Beenden von Diensten auf dem neuen „Master“ der Rechnergruppe.

„heartbeat“ baut dazu einen Cluster über alle ihm zugeteilten Server auf, der an eine virtuelle IP und einen Hostnamen gebunden wird. Die Software stellt sicher, dass die IP jeweils genau an einen Server aus dem Cluster vergeben wird, der Netzkonnektivität besitzt und auf dem alle spezifizierten Prozesse laufen beziehungsweise sich fehlerfrei starten lassen.

#### **4.5.2 fake**

Diese Software, die unter [fak04] bereitgestellt wird, unterstützt den Aufbau eines hochverfügbaren Systems dadurch, dass damit die IP eines ausgefallenen Rechners übernommen werden kann. Dies geschieht mit Hilfe von ARP-Spoofing, also in dem die ARP-Anfragen an die fremde IP durch den Rechner beantwortet werden. Selbst beschreibt sich das Projekt wie folgt:

Fake has been designed to switch in backup servers on a LAN. In particular it has been designed to backup Mail, Web and Proxy servers during periods of both unscheduled and scheduled down time.

Fake allows you to take over the IP address of another machine in the LAN by bringing up an additional interface and making use of ARP spoofing. The additional interface can be either a physical interface or an IP alias.

Fake is easily configurable and is designed to enable automated invocation via systems such as Mon that monitor the availability of servers. Fake can also be used in conjunction with load balancing mechanisms such as The Linux Virtual Server.

### 4.5.3 mon

Wird also, wie von „fake“ ein dienstspezifisches Monitoring benötigt, verweist unter anderem auch die Entwicklergemeinde von „heartbeat“ auf das Werkzeug „mon“ (siehe dazu [mon04]), welches ein Framework darstellt, um mittels Testskripten einerseits die Verfügbarkeit eines Dienstes zu testen und andererseits im Fehlerfall eine Alarmierung durch gesonderte Anweisungsfolgen auszulösen.

Mit Hilfe eigener Skripte kann auch dieses Werkzeug genutzt werden, um Fehler oder Ausfälle festzustellen und darauf geeignet zu reagieren.

### 4.5.4 failover

In dieser Software vereinigen sich wieder die Komponenten der Überwachung und Reaktion. Die auf [fai04a] von Andreas Müller bereitgestellte Software basiert auf dem Konzept zweier Daemons, die eine gegenseitige Überwachung zweier Rechner ermöglichen.

Dabei signalisiert der eine Daemon regelmäßig seine Verfügbarkeit und der andere Daemon nimmt die Signalisierungen des anderen Servers entgegen. Fehlt irgendwann diese Signalisierung des zweiten Rechners, werden Skripte angestoßen, die entsprechende Reaktionen auslösen sollen.

### 4.5.5 keepalived

Ein Produkt, welches speziell für den Einsatz am LVS-Server gedacht ist, wird unter [kee04] gehostet.

Dieses baut eine Überwachung über einen Pool von LVS mittels VRRP auf und informiert beteiligte Server des Pools über Ausfälle anderer Server. Ausserdem unterstützt es ein automatisches Failover der Director-Rolle im LVS.

Das Projekt vereinigt gute Ideen für die Überwachung und Ereignisreaktion, ist aber leider ausschließlich auf das LVS-Projekt anwendbar. Die Idee einer VRRP-Umsetzung findet sich aber im folgenden Produkt wieder.

#### 4.5.6 VRRP<sub>d</sub> (ehemals failover<sub>d</sub>)

Das auf [fai04b] beheimatete Projekt stellt eine Software bereit, die ein automatisches Fail-over auf Grundlage des in [vrr98] beschriebenen VRRP (Virtual Redundant Router Protocol) ermöglicht.

Dabei werden mittels Multi- oder Broadcast Meldungen der bereitstehenden Server im Netz verteilt und aus den beteiligten Rechnern ein Master gewählt. Fällt dieser aus, findet automatisch die Bestimmung eines neuen Masters aus der Gruppe der sich meldenden Backup-Server statt. Der Master erhält eine virtuelle MAC und IP, über die der Service-Traffic läuft und für die sichergestellt wird, dass sie jederzeit erreichbar ist.

#### 4.5.7 Big Brother

Gegenüber den gerade genannten Produkten wird dieses hier bereits in großem Umfang im Rechenzentrum der TU Chemnitz für das Infrastruktur-Monitoring eingesetzt, was ihm einen Vorteil verschafft, da es zusätzlich auch zum internen Monitoring der Serverplattform genutzt werden kann.

Die Software ist skriptbasiert. In festgelegten Abständen wird mittels eines Cronjobs die Ausführung von Shellskripten angestoßen, deren Ergebnisse an einen zentralen Monitoring-Server geschickt werden, die aber eben auch die Reaktion auf die Feststellungen auslösen. Dadurch ist es notwendig, die Reaktion auf einen Ausfall eines Servers durch das Shellskript des Anderen auslösen zu lassen.

Beispielsweise kann auf ein Nichterreichen des HTTP-Services auf *bms1* durch *bms2* zu der Überzeugung führen, *bms1* sei ausgefallen und mittels „ifconfig“ zur Übernahme der IP *bms1* durch *bms2* führen.

Da auch alle anderen vorgestellten Produkte zum Monitoring ihre Randbedingungen haben und eigentlich nur „heartbeat“ durch seine kontinuierliche Überwachung Vorteile brächte, aber eben auch den Aufwand der Betreuung eines zusätzlichen Produktes, erachte ich es als ausreichend, den Einsatz von BigBrother um die gewünschte Funktionalität zu erweitern.

## 5 Der Bildungsmarktplatz Sachsen – eine Empfehlung zur Serverstruktur

In der Folge nun die Essenz der Betrachtungen, die sich zu einer sinnvollen Lösung für den Bildungsmarktplatz Sachsen kombinieren lassen, um den größtmöglichen Nutzen daraus zu ziehen.

### 5.1 Konzept zur Lastverteilung

Hinsichtlich der Frage der Lastverteilung ergibt sich relativ schnell anhand der genannten Alternativen, dass das Konzept des RoundRobin-DNS die günstigste Lösung darstellt. Es verbindet einen geringen administrativen Aufwand mit ausreichend Nutzen. Dieser kann aufgrund des geschickten Einsatzes von IPs auch im Falle eines Ausfalls ohne Probleme weiterarbeiten, ohne am DNS-System an sich Eingriffe vornehmen zu müssen. Das RoundRobin wird über den Namen *www.bildungsmarkt-sachsen.de* angesetzt und dieser wird auf die IPs abgebildet, die den Namen *bms1.hrz.tu-chemnitz.de* und *bms2.hrz.tu-chemnitz.de* zugeordnet sind.

Dabei sind die Ideen der Hochverfügbarkeit derart, dass im Falle eines Ausfalls dafür Sorge getragen wird, dass die IPs, die den Namen *bms1* und *bms2* zugeordnet sind, stets erreichbar bleiben. Im Falle eines Ausfalls übernimmt der dann einzig funktionstüchtig gebliebene Server beide IPs und alle Funktionen.

Gegen die Alternative der dienstbezogenen Lastverteilung spricht der Aspekt, dass hier genau ein externer Dienst verteilt werden soll.

Hingegen die Alternativen der Lastverteilung mittels virtuellem Server wären anwendbar, haben aber einen deutlich höheren administrativen Aufwand in der Einrichtung und einer komplexen Konfiguration. Hinzu kommt, dass eine weitere IP, nämlich die virtuelle, benötigt würde und der Lastverteiler einen neuen Single Point of Failure darstellt. Die Lastverteilung mittels Switch stellt ebenfalls einen höheren Kostenfaktor dar, da die Hardware, die so etwas unterstützt, deutlich teurer ist, als die so benötigte. Zur Lastverteilung mit einem LVS würde eine weitere Maschine notwendig werden, die diese Funktion übernimmt und in ihrer zentralen Funktion überwacht werden muss sowie weiteren administrativen recht umfangreichen Aufwand darstellt.

Damit die Lastverteilung die Arbeit nicht negativ beeinflusst, müssen ja beide Maschinen einen identischen Datenbestand aufweisen. Dazu dient der ZEO-Server, der eine einheitliche Sicht auf die ZODB von beiden Instanzen aus gewährleistet und koordiniert. Die restlichen Daten befinden sich im PostgreSQL-Datenbankservers, der für beide Maschinen per Netz erreichbar ist.

Im Zusammenhang mit der PostgreSQL-Datenbank sei zur Lastverteilung noch erwähnt, dass zwar, wie bei „pgpool“ angedeutet, eine Lastverteilung möglich, aber recht schwierig ist. Vorallem in der Kombination von Lastverteilung mit einer Sicherung der Datenkonsistenz über mehrere Systeme stellen sich Effekte dar, die widerspüchliche Anforderungen aufzeigen.

Wie im „pgpool“ ist nur bei ausschließlich lesendem Zugriff und einer davon getrennten Wahrung der Datenkonsistenz eine einfache Lösung der Lastverteilung möglich. Eine volle Lastverteilung wäre nur in einer Kombination denkbar, in der sowohl ein Werkzeug die Ausführung abhängiger Transaktionen geeignet serialisiert, eine passende Verteilung über die Datenbankservers findet und ein geeigneter Mechanismus gegeben ist, der eine gegenseitige Replikation mehrerer Datenbanken sichert. Im Kontext der verteilten Datenbanken wird dies meist auch nur durch Einsatz eines Servers erreicht, der das Scheduling der Transaktionen und die volle Verteilung kontrolliert.

Für die hier geplante Applikation ist daher von einem derart komplexen Systemaufbau abzuraten.

## 5.2 Caching

Aufgrund der Integration vieler Funktionen spezieller Caching-Produkte in die Standard Cache Manager (Accelerated HTTP Cache und RAM Cache) sowie die Nutzungsszenarien des Projektes machen den Einsatz vieler zusätzlicher Cachingprodukte nicht lohnenswert. Außerdem ist zu bedenken, dass ein Objekt in der ZOPE-Hierarchie jeweils nur einem Cachetyp, viele sogar überhaupt nicht, zuzuordnen ist. Damit müssen die Vorteile des Produkts in der Folge alle Vorteile anderer Produkte aufwiegen.

So fielen die Cache Controlled ZSQL Methods aus der Betrachtung, da der erreichte Speedup gleich Null war und ZSQL-Methoden darüber hinaus nicht benutzt werden. Die Produkte des externen Filesystem-Cachings, sei es der FS-Cache oder ZOPE-Proxy-Cache, waren nicht funktionstüchtig oder konnten ihre Funktionstüchtigkeit nicht unter Beweis stellen.

Es blieb also in der Gegenüberstellung der beiden Standard Cache Manager. Dabei zeigte sich aufgrund der Messungen mittels des Apache Benchmarks, dass der RAM-Cache gute, aber bei nicht verändernden Seiten der HTTP-Cache unschlagbare Speedups aufwies. Ebenfalls konnte der HTTP-Cache seine Vorteile bei den Messungen über ein entferntes Netz ausspielen. Aufgrund der ausschließlichen Header-basierten Kontrolle kommt es allerdings

bei Änderungen am Objekt zur Auslieferung veralteter Seiten.

Daher ist dieser Cache-Typ für Seiten mit sich ändernden Inhalten eher ungeeignet. Dort kann der RAM-Cache seine Vorteile ausspielen, der auch Teile einer gerenderten Seite einlagern und zurückliefern kann. Außerdem eben solange aktuell ist, wie das Objekt nicht verändert wird, aber über dessen Änderung informiert und invalidiert wird.

Also wäre ein Verknüpfung von Skripten an den RAM-Cache und von Portal-Skin-Bestandteilen, wie Bildern und Stylesheets, eine sinnvolle Kombination, sollten die gewünschten Objekte überhaupt mit einem Cache assozierbar sein.

Ergänzend dazu bleibt die auf [apc04] beschriebene Idee, Inhalte auf der Grundlage der Dokumentarten anhand der Content-Type-Header im dem ZOPE-Server vorgelagerten Apache-Cache durchzuführen. Dabei bieten sich vor allem Grafiken an, die einen recht statischen Anteil am Webauftritt darstellen. Nur im Falle einer Überarbeitung des Corporate Designs und der CMS-Skins käme es zu Aktualisierungen, die ein Invalidieren von gecachelten Grafiken nach sich ziehen könnte. Dadurch könnte nicht nur eine Beschleunigung der Antworten des Servers erreicht werden, sondern auch eine Entlastung des ZOPE-Servers.

## 5.3 Hochverfügbarkeit

Als Grundlage eines Hochverfügbarkeitskonzeptes wird der Einsatz von 5 DNS-Namen und 4 IPs vorgeschlagen. Dabei haben beide Maschinen ihre, ihnen fest zugeordneten Namen *clara.hrz.tu-chemnitz.de* und *richard.hrz.tu-chemnitz.de* und die zugehörigen IPs. Diese werden automatisch durch die Maschinen beim Start belegt und dienen der internen Administration der Plattform.

Weiterhin gibt es die beiden Namen *bms1.hrz.tu-chemnitz.de* und *bms2.hrz.tu-chemnitz.de*, die manuell an die Maschinen vergeben werden und die implizit die Rollen des Masters und Slave hinsichtlich der Datenreplikation vergeben.

Der letzte Name ist *www.bildungsmarkt-sachsen.de*, über den eine Lastverteilung per Round-Robin DNS angewendet wird und der der einzig extern genutzte Name ist.

### 5.3.1 Überwachung und IP-Übernahme

Um die Ausfallsicherheit der Plattform adäquat zu erhöhen, ist eine Überwachung mittels Big Brother angedacht, wobei die Server sich ebenfalls gegenseitig monitoren. Im Normalbetrieb hat der eine Rechner die IP und den Namen *bms1* und der andere das Tupel zu *bms2*. Dabei stellt *bms1* neben einer ZOPE-Instanz den ZEO-Server und den Datenbankserver bereit, ist also hinsichtlich der Replikation der Datenbank-Daten der Master.

Durch den Effekt, dass mittels Skripten gesichert wird, dass diese beiden IPs immer erreichbar sind und über *bms1* immer der ZEO- und Datenbankserver erreichbar bleiben, können die Konfigurationen der ZOPE-Instanzen und verwendeten Python-Skripte statisch auf den

entsprechenden Namen *bms1* erfolgen.

Die Sicherung der Erreichbarkeit beider IPs wird, wie im Folgenden beschrieben, erreicht: Kommt es nun zu einem Ausfall von *bms2* wird dies beim nächsten Lauf des BigBrothers durch *bms1* festgestellt, da dann ein überwachter Dienst nicht mehr erreichbar ist. Ein Skript, welches durch den BigBrother ausgelöst wird, sorgt dafür, dass *bms1* zusätzlich die IP von *bms2* mittels *ifconfig* übernimmt. Weiter ist in diesem Falle kein Eingriff notwendig, da alle Serverfunktionalität bei *bms1* liegt und funktionsbereit ist. Somit werden in der Folge bis zum Failback alle Anfragen von *bms1* bearbeitet.

Fällt hingegen *bms1* aus, ist die weitere Funktionalität des ZEO- und Datenbank-Servers zu gewährleisten. Dazu wird nach Feststellung des Ausfalls jeweils eine Serverinstanz auf *bms2* durch die Übernahme der IP *bms1* aktiviert. Die Dienste laufen bereits auf den Maschinen, sind sie doch zur Replikation der Daten notwendig, welche im Normalbetrieb von *bms1* auf *bms2* stattfindet, damit beide Instanzen einen adäquaten Datenbestand aufweisen.

## 5.3.2 Datenbank-Replikation

### ZODB-Replikation

Bei der Replikation der ZODB stehen sich ja die Mechanismen mittels ZSyncer, per „*repozo.py*“ oder *rsync* auf Fileebene gegenüber.

Für die Variante mit dem ZSyncer-Produkt spricht, dass es voll in ZOPE integriert ist und einen eleganten Abgleich ermöglicht. Problematisch stellt sich dar, dass dazu eine ZOPE-Instanz gestartet sein muss, die auf die Standby-ZEO-Instanz zugreift und darüber synchronisiert. Also eine Instanz muss auf einen ZEO-Server unter der IP *bms2* zugreifen, während die Produktiv-Instanzen auf einen ZEO-Server *bms1* zugreifen. Sinnvoll wäre die Initialisierung dieser Instanz auf *bms1*, da damit bei dessen Ausfall die Replikation automatisch endet.

Da die Variante allerdings einigen Aufwand mit Skripten mit sich bringt, die die Webseite des Zsyncer-Moduls laden, parsen und gegebenenfalls die Synchronisation per Webformular auslösen, und in einem frühen Test Probleme mit manchen Objekttypen auftraten, die später nicht reproduzierbar waren, aber im Workbetrieb vermieden werden sollten, wäre ein Probeinsatz von „*repozo.py*“ sinnvoll. Dabei könnte auf dem Slave ein Skript zum Einsatz kommen, welches mittels „*ls*“ und „*lsif*“ regelmäßig überprüft, ob ein neues Backup vom Master abgelegt und fertig kopiert wurde. In der Folge kann dieses File dann mittels „*repozo.py*“ in die ZODB des Standby-ZEO-Servers eingespielt werden. Dazu wäre keine zusätzliche Instanz notwendig, und aufgrund der inkrementellen Unterstützung des Werkzeugs ist sogar ein kurzer Zyklus zwischen den Updates denkbar. Bei einem Ausfall des Masters muss sicherheitshalber der „Daemon“-Prozess beendet werden, falls nur einzelne Dienste des alten Masters nicht erreichbar sind oder waren und dadurch „neue“ Backupdateien erstellt wer-

den. Als Medium der Übertragung ist entweder ein Netzlaufwerk denkbar oder ein Kopieren mittels `rsync`, welches vom Master nach seinem per Cronjob angestoßenen Backup gestartet wird.

Als dritte Wahl stünde dann das Kopieren auf Filesystemebene mittels `rsync`, wobei es gleiche Bedingungen wie der Einsatz von „`repozo.py`“ hat, eben nur die Vorteile der vollen ZOPE-Integration nicht aufweist, aber vorm Kopieren auch nicht erst einen Abzug erstellen lassen muss.

### PSQL-Replikation

Ganz ähnlich verhält es sich beim Replizieren der PostgreSQL-Datenbank, die mittels des Tools „`Slony-1`“ durchgeführt wird. Günstig hierbei sind die erwähnten Vorteile, wie ein dynamisches Switchover und die Wahrung der Datenkonsistenz auch bei kurzzeitigen Problemen der gegenseitigen Erreichbarkeit. Im Falle eines Gesamtausfalls eines Rechners kommt außerdem hinzu, dass eine Replikation auf Basis der RechnerIPs, also *clara* und *richard*, automatisch aufgrund deren Nichterreichbarkeit dafür sorgt, dass die weitere Replikation unterbunden wird. Kommt es allerdings nur zu einem Teilsystemausfall muss bei Übernahme der IP *bms1* und damit der Masterrolle dafür Sorge getragen werden, dass Effekte durch eventuelle Replikationen vom alten Master auftreten. Es muss also auf dem dann neuen Master der Dienst, der die Backups des alten Masters einspielte, eingestellt werden.

Dazu ist es entweder möglich, den `Slony-1`-Dienst auf der Maschine bis zum Failback zu beenden oder aber diesem Rechner die Rolle des Masters mittels des auf [slo04] beschriebenen Switchover zu übertragen.

### 5.3.3 Hinweise hinsichtlich eines Failback

Kam es zu einem Ausfall in der Plattform, ist sicher zu stellen, dass keine Datenreplikation von der ausgefallenen, inzwischen datenseitig veralteten Maschine auf die aktive Plattform stattfindet. Daher sollte die Replikationsdienste an die IPs oder Namen *bms1* und *bms2* gebunden werden und deren Vergabe nach einem Ausfall gewissenhaft und manuell stattfinden. Die Maschinen booten mit ihren Namen und IPs als *clara* und *richard*, und erst nach einer Überprüfung sollte einem ausgefallenen Rechner die IP *bms2* wieder zugeordnet werden. In der Folge kann dann dieser Knoten *bms2* in den `Slony-1`-Cluster als Slave eingebunden und die Skripte zur Erstellung von ZODB-Backups durch *bms1* und deren Einspielung durch *bms2* aktiviert werden.

## 5.4 Mögliche Zustände der Plattform

Zustand	bms1	bms2	Erläuterungen
Normal (siehe Abbildung 5.1)	<ul style="list-style-type: none"> <li>• ZOPE-Server hinter Apache-Frontend</li> <li>• ZEO-Server</li> <li>• PostgreSQL-Server</li> <li>• Replikation durch <code>repo-zo.py</code> und <code>Slony-1</code></li> </ul>	<ul style="list-style-type: none"> <li>• ZOPE-Server hinter Apache-Frontend</li> </ul>	Die vom DNS verteilten Anfragen gehen an <i>bms1</i> und <i>bms2</i> , werden dort durch den ZOPE-Server bearbeitet, der dazu auf die Datenbanken auf <i>bms1</i> zugreift. Diese Datenbanken werden von <i>bms1</i> aus auf <i>bms2</i> repliziert.
Ausfall <i>bms2</i> (siehe Abb. 5.2)	<ul style="list-style-type: none"> <li>• ZOPE-Server hinter Apache-Frontend</li> <li>• ZEO-Server</li> <li>• PostgreSQL-Server</li> <li>• <i>Übernahme der IP bms2</i></li> </ul>	<i>ausgefallen</i>	Durch den Ausfall übernimmt <i>bms1</i> die IP <i>bms2</i> . Die Replikation der Datenbanken wird eingestellt, solange kein Slave vorhanden ist. Alle benötigten Dienste laufen auf <i>bms1</i> , wodurch es zu keinem Service-Ausfall kommt. Die durch den DNS-Service aufgeteilten Anfragen werden auf <i>bms1</i> indirekt wieder zusammengeführt.
Ausfall <i>bms1</i> (siehe Abb. 5.2)	<i>ausgefallen</i>	<ul style="list-style-type: none"> <li>• ZOPE-Server hinter Apache-Frontend</li> <li>• ZEO-Server</li> <li>• PostgreSQL-Server</li> <li>• <i>Übernahme der IP bms1</i></li> </ul>	Durch den Ausfall von <i>bms1</i> kam es zum Ausfall der Datenquellen, PostgreSQL- und ZEO-Server, deren Dienste nun durch die replizierten Datenbanken auf <i>bms2</i> übernommen werden. Ebenso wie die IP <i>bms1</i> . Dadurch sind nun auf <i>bms2</i> alle Dienste bereitgestellt und die IPs der Plattform vereint. Durch diesen Mechanismus ist der Ausfall kompensiert.

Zustand	bms1	bms2	Erläuterungen
Failback eines ausgefallenen Rechners	<ul style="list-style-type: none"> <li>• ZOPE-Server hinter Apache-Frontend</li> <li>• ZEO-Server</li> <li>• PostgreSQL-Server</li> <li>• <i>Replikation durch repo-zo.py und Slony-1 neu initialisieren</i></li> </ul>	<p><i>Dies ist der neugestartete Rechner</i></p> <ul style="list-style-type: none"> <li>• ZOPE-Server hinter Apache-Frontend</li> </ul>	<p>Da nach einem Failover jeweils der verbliebene Rechner die Funktion des Masters übernommen hat, ist es sinnvoll den neugestarteten Rechner im Rahmen des Failbacks als Slave, also <i>bms2</i> einzubinden. Dazu wird der ZOPE- und Apache-Dienst auf diesem Rechner gestartet und die ZOPE-Instanzen mit dem ZEO-Server verbunden. Danach kann ihm die IP <i>bms2</i> übertragen. Nach dieser Eingliederung wird das Replizieren mit einem Fulldump neu initialisiert. Im Anschluss ist der Normalbetrieb wieder erreicht.</p>

Tabelle 5.1: Zustandsfälle der Plattform

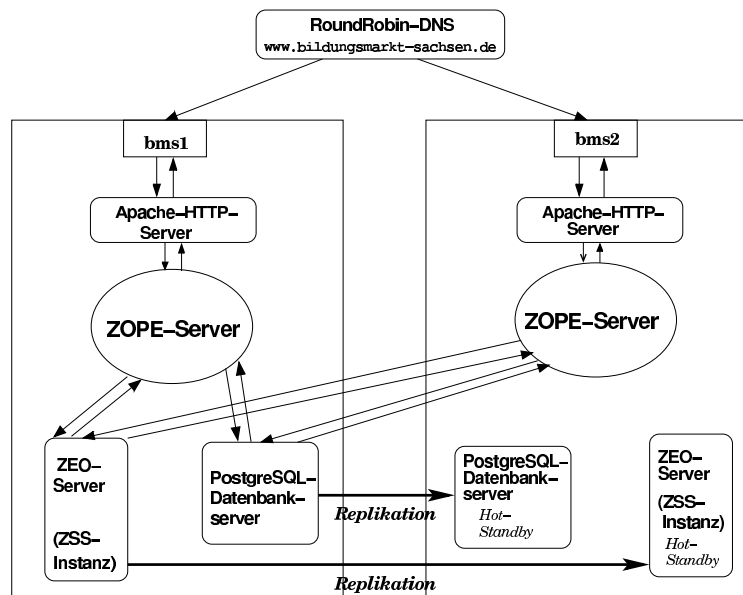


Abbildung 5.1: Gesamtvorschlag zur Lösung

Im Falle eines geplanten Wartungsfensters bieten sich aufgrund der Plattform ebenfalls die Möglichkeit, dieses für den Endnutzer transparent durchzuführen. Dazu wird die Replikation eingestellt beziehungsweise, sollte der zu wartende Rechner *bms1* sein, *bms2* im Slony-1 zum Master deklariert. Danach kann die IP des zu wartenden Servers an den zweiten übertragen werden.

Damit ist der Server aus der Plattform gelöst und kann nun uneingeschränkt zur Wartung herangezogen werden. Nach dieser wird ganz genau nach Verfahren zur Einbindung im Rahmen eines Failbacks vorgegangen und der gewartete Rechner als *bms2* und damit als Slave in die Plattform wieder aufgenommen.

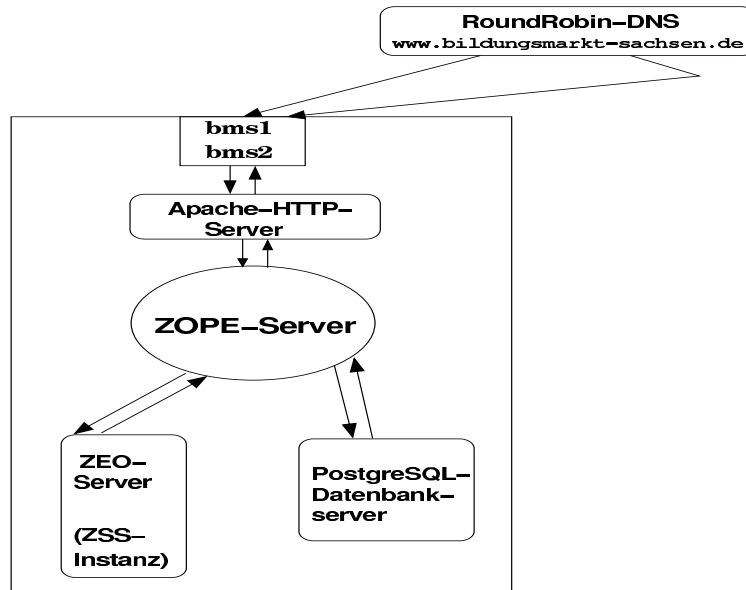


Abbildung 5.2: Betriebsweise nach einem Ausfall

## 6 Fazit

In der Gesamtbetrachtung der Ergebnisse der vorliegenden Arbeit ist zusammenfassend festzustellen, dass durch die vorgeschlagene Kombination der eingesetzten Werkzeuge die Möglichkeit zur Steigerung der Performanz besteht. Dabei haben die Aspekte des Caching ebenso Einfluss, wie die Resultate der Lastverteilung, wodurch eine Verkürzung der Antwortzeiten zu erwarten ist.

Hinsichtlich der Aspekte der Hochverfügbarkeit ist anzumerken, dass die Replikation verbunden mit den Konzepten zur Übernahme der IPs mittels BigBrother und „repozo.py“ beziehungsweise „Slony-1“ eine akzeptable Steigerung und Sicherungserhöhung darstellen. Allerdings besteht gerade bei der ZODB-Replikation aufgrund der nur periodischen Replikation, also nicht von Änderungen am Datenbestand gesteuert, Verbesserungspotential, welches mit derzeitigen freien Werkzeugen nicht realisierbar ist. Ein Produkt, welches dieser Anforderung gerecht wird, scheint das kommerzielle ZRS (Zope Replication Service) zu sein.

Bezüglich der Überwachung und IP-Übernahme wäre eventuell über den Einsatz von „heartbeat“ nachzudenken, da damit wahrscheinlich eine schnellere Reaktionszeit zu erwarten ist. Da BigBrother ebenfalls nur periodisch läuft, kann es zu einem Zeitfenster zwischen zwei Läufen kommen, in denen ein Ausfall bereits geschehen, aber unbemerkt ist. Dem ist entgegen zu stellen, dass BigBrother bereits stark im Rechenzentrum der TU Chemnitz verbreitet ist und somit auf den Maschinen mehrere Monitoring-Systeme eingesetzt, gewartet und auf funktionale Korrektheit geprüft werden müssen.

Solange allerdings die Rahmen des Service Level Agreements eine kürzere Ausfallzeit zulassen, sollte im Sinne der Minimalität vielleicht darauf verzichtet werden. BigBrother läuft im 5-Minuten-Intervall, was somit als maximale Ausfallzeit auftreten kann.

Die Lösung ist dahingehend skalierbar, dass der Einsatz weiterer Server mit diesem Konzept denkbar und realisierbar ist. Die Mechanismen der Replikation sind ohne Weiteres auf eine größere Zahl von Klienten erweiterbar, im Slony sogar primär vorgesehen. Das RoundRobin im DNS ist einfach zu erweitern und ermöglicht damit, ebenso wie der ZEO-Server, auch den Einsatz weiterer Server als Slaves und ZOPE-Frontend.

Bedenkenswert bei einer Erweiterung ist die Problematik, wie im Falle eines Ausfalls der neue Master bestimmt wird und welcher Slave die IP und Aufgaben des bisherigen Masters übernimmt und dessen Ausfall überbrückt. Aber schon die Implementierung eines einfachen Wahlverfahrens unter den Verbliebenen schafft dort Abhilfe.

## Literaturverzeichnis

- [apa04] *Apache HTTP Server Project*, Aug. 2004,  
<http://httpd.apache.org>.
- [apc04] *Content-based Caching im Apache*, Nov. 2004,  
<http://zope.org/Members/nexedi/accelerate-zope.stx>.
- [ape04] *Ape - Adaptable Persistence Engine*, Aug. 2004,  
<http://hathaway.freezope.org/Software/Ape>.
- [ccz04] *Cache Controlled ZSQL Methods*, Febr. 2004,  
<http://zope.org/Members/pupq/CacheControlledZSQLMethods>.
- [cis04] *Cisco IOS Server Load Balancing*, Okt. 2004,  
[http://www.cisco.com/en/US/products/sw/iosswrel/ps1833%/products\\_feature\\_guide09186a0080080fd2.html#1024373](http://www.cisco.com/en/US/products/sw/iosswrel/ps1833%/products_feature_guide09186a0080080fd2.html#1024373).
- [cmf04] *CMF Webseite*, Nov. 2004,  
<http://http://zope.org/Products/CMF>.
- [dD97] Wissenschaftlicher Rat der Dudenredaktion (Hg.): *Duden - Das Fremdwörterbuch*, Bibliographisches Institut & F.A. Brockhaus AG, Mannheim, Wien, Zürich, 1997, ISBN 3-411-04056-4.
- [fai04a] *failover Homepage*, März 2004,  
<http://failover.othello.ch/>.
- [fai04b] *failoverd Homepage*, Nov. 2004,  
<http://ps-ax.com/failoverd/>.
- [fak04] *fake Homepage*, Juni 2004,  
<http://www.vergenet.net/linux/fake/>.
- [fsc02] *Filesystem Cache Manager*, Nov. 2002,  
<http://zope.org/Members/andym/FSCacheManager>.
- [kee04] *HealthChecking for LVS & HighAvailability*, Nov. 2004,  
<http://www.keepalived.org/>.

- [kon03a] *Kontentor 2.0 Handbuch*, 2003,  
[http://www.kontentor.de/main/download/HF\\_sections/  
content/KontentorUserDocumentation.pdf](http://www.kontentor.de/main/download/HF_sections/content/KontentorUserDocumentation.pdf).
- [kon03b] *Kontentor CMS*, 2003,  
<http://www.kontentor.de>.
- [lin04] *High-Availability Linux Project*, Nov. 2004,  
<http://linux-ha.org>.
- [lvs04] *Technik des Linux Virtual Servers*, Okt. 2004,  
<http://www.linuxvirtualserver.org/how.html>.
- [MB01] BA Ravensburg Michael Braig: *Entwicklung eines Leitfadens zur Umsetzung von Hochverfügbarkeitskonzepten am Beispiel Cooper Standard Automotive*, 2001,  
<http://www.wissen24.de/vorschau/7682.html>.
- [McK04] Andy McKay: *Profiling, Benchmarking and Caching in Plone*, Nov. 2004,  
[http://www.clearwind.ca/talks/  
profiling\\_and\\_caching.pdf](http://www.clearwind.ca/talks/profiling_and_caching.pdf).
- [mon04] *mon Homepage*, Okt. 2004,  
<http://www.kernel.org/software/mon/>.
- [MP02] Amos Latteier u.a. Michel Pelletier: *Das ZOPE-Buch*, Markt + Technik Verlag, München, 2002, ISBN 3-8272-6194-5.
- [mtc04] *marketingterms Cache Definition*, 2004,  
<http://www.marketingterms.com/dictionary/caching/>.
- [per01] *ZOPE Proxy Cache Manager*, Nov. 2001,  
<http://www.zope.org/Members/xgwsbae/ZopeProxyCacheManager>.
- [plo04] *Plone Webseite*, Aug. 2004,  
<http://www.plone.org/>.
- [psq04] *PostgreSQL Webseite*, Sept. 2004,  
<http://www.postgresql.org/>.
- [pyt04] *Python Webseite*, Aug. 2004,  
<http://www.python.org/>.
- [ram01] *RAM Cache Manager with Aging*, Juli 2001,  
[http://zope.org/Members/xgwsbae/  
RamCacheManagerWithAge](http://zope.org/Members/xgwsbae/RamCacheManagerWithAge).

- [Ric03] Stephan Richter (Hg.): *Content Management mit ZOPE*, Software & Support Verlag, Frankfurt, 2003, ISBN 3-935042-44-2.
- [Sch00] Michael Schwingel: *Gigabit Ethernet Praktikum*, Nov. 2000, tU Chemnitz.
- [slo04] *Slony-1 Projektseite*, Okt. 2004,  
<http://gborg.postgresql.org/project/slony1/projdisplay.php>.
- [sni04a] *marketingterms Cache Definition*, 2004,  
<http://www.snia.org/education/dictionary/l/>.
- [sni04b] *SNIA Cache Definition*, 2004,  
<http://www.snia.org/education/dictionary/c/>.
- [sni04c] *SNIA High availability Definition*, 2004,  
<http://www.snia.org/education/dictionary/h/>.
- [sql04] *SQL-Relay Project Page*, Nov. 2004,  
<http://sqlrelay.sourceforge.net/>.
- [squ04] *Squid Web Proxy Cache*, Juli 2004,  
<http://www.squid-cache.org/>.
- [srf04] *SQL-Relay FAQ*, Nov. 2004,  
<http://sqlrelay.sourceforge.net/sqlrelay/faq.html>.
- [std00] *ZOPE Standard Cache Manager*, Dez. 2000,  
<http://zope.org/Members/hathawsh/StandardCacheManagers>.
- [tec04] *Techtarget Cache Definition*, Dez. 2004,  
[http://searchwebservices.techtarget.com/sDefinition/0,,sid26\\_gci211728,00.html](http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci211728,00.html).
- [vrr98] *RFC 2883 - Virtual Router Redundancy Protocol*, Apr. 1998,  
<ftp://ftp.ietf.org/rfc/rfc2338.txt>.
- [wpc04] *Wikipedia Cache Definition*, Dez. 2004,  
<http://de.wikipedia.org/wiki/Cache>.
- [wpl04] *Wikipedia Lastverteilung Definition*, Nov. 2004,  
<http://de.wikipedia.org/wiki/Lastverteilung>.
- [wpp04] *Wikipedia Performanz Definition*, Sept. 2004,  
<http://de.wikipedia.org/wiki/Performanz>.
- [zop04] *Webseite des ZOPE Projekts*, Juli 2004,  
<http://www.zope.org/>.

- [zyt04] *HOWTO - Configure Round Robin and Load Balancing*, 2004,  
<http://www.zytrax.com/books/dns/ch9/rr.html>.

# Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Chemnitz, den 22. Dezember 2004

---

Damaschke Marko

