Diplomarbeit

Datenbankabstraktion im CATACOMB WebDAV Server mit mod_dbd und exemplarischer Implementierung

Bernd Sydow
Hochschule Aalen

DLR

Datenbankabstraktion im Catacomb WebDAV Server mit mod_dbd und exemplarischer Implementierung

Diplomarbeit

von

Bernd Sydow
geboren am 16. Februar 1984 in Aalen
Matrikelnummer 22604

31. März 2008

Betreuer:
Prof. Dr. R. Werthebach (HTW Aalen)
Dipl. Math. A. Schreiber (DLR)
Dipl. Inf. (FH) M. Litz (DLR)

Hochschule Aalen
Fachbereich Elektronik und Informatik

Erklärung

Ich versichere hiermit, die vorliegende Diplomarbeit mit dem Thema

Datenbankabstraktion im Catacomb WebDAV Server mit mod_dbd und exemplarischer Implementierung

selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Köln, den 31. März 2008

Zusammenfassung

Häufig kommt in Software Projekten früher oder später die Frage auf, wie man unabhängiger von einer Datenbank eines Herstellers wird. In den seltensten Fällen ist es gewünscht oder notwenig auf die Datenbank eines Herstellers zu setzen. Eine Applikation unabhängig von einer Datenbank zu gestalten, bietet einige Fallstricke.

Der Catacomb WebDAV Server ist bei seiner Entstehung auf eine spezifische Datenbank hin entwickelt worden. Ziel dieser Diplomarbeit ist den Catacomb WebDAV Server Datenbankunabhängig zu machen und eine exemplarische Implementierung zu erstellen.

Abstract

Often, there appears in software projects sooner or later the question, how to be more independent of a database from the manufacturers. In the rarest cases, it is desired or necessary to focus on a database of the manufacturer. Designing an application independent from an database, causes many pitfalls.

The Catacomb WebDAV Server was developed in its development to use a certain database. The intention of this diploma thesis was, to make the CatacombWebDAV Server database independent and to build an example implementation.

Inhaltsverzeichnis

1 Einführung

1.1 Einleitung

Das Deutsche Zentrum für Luft- und Raumfahrt (DLR) ist die zentrale Forschungseinrichtung der Bundesrepublik Deutschland für die Bereiche Luft- und Raumfahrt. Über die eigene Forschung hinaus ist das DLR als Raumfahrtagentur im Auftrag der Bundesregierung für die Planung und Umsetzung der deutschen Raumfahrtaktivitäten zuständig.

Die Einrichtung Simulations- und Softwaretechnik ist innerhalb des DLR für Bereitstellung, Entwicklung und Forschung auf den Gebiet Innovative Software-Engineering-Technologien zuständig. Die derzeitigen Themenschwerpunkte sind die komponentenbasierte Softwareentwicklung für verteilte Systeme, Software für eingebettete Systeme und Software-Qualitätssicherung.

Die Einrichtung Simulations- und Softwaretechnik entwickelt mit dem Datafinder ein Programm zur Verwaltung technisch-wissenschaftlicher Daten. Dieses Programm verwendet als Datenspeicher einen Server der das WebDAV Protokoll spricht. Bisher können der kommerzielle Tamino Server und Catacomb als Server eingesetzt werden.

Der Catacomb Server ist Open Source und wird aktiv vom DLR weiterentwickelt. Catacomb unterstützt derzeit nicht den vollen Funktionsumfang von WebDAV.

1.2 Aufgabenstellung

Der Catacomb WebDAV Server verwendet zum Speichern der Daten eine Relationale Datenbank. Derzeit ist es die Datenbank MySQL. In Zukunft sollen auch Relationale Datenbanken anderer Hersteller als Backend verwendet werden können.

Im Rahmen dieser Diplomarbeit soll der Catacomb WebDAV Server vollständig von der Datenbank MySQL abstrahiert werden. Neben der Abstraktion der Datenbank ist auch auf die Transaktionsfähigkeit zu achten.

1.3 Gliederung

Kapitel 2 Hier werden die grundlegenden Sachverhalte vermittelt, die für das Verständnis der Arbeit erforderlich oder förderlich sind. Dies beinhaltet WebDAV im Allgemeinen, die Apache 2 Architektur und Datenbanken.

Kapitel 3 In diesem Kapitel werden die Datenbankmodelle des Catacomb Servers vorgestellt.

Kapitel 4 Zeigt, wo welche Änderungen an Catacomb und der Datenbank durchzuführen sind, um Datenbankunabhängigkeit und Transaktionsfähigkeit zu erreichen.

Kapitel 5 Beschreibt wie die geplanten Änderungen aus dem Kapitel zuvor konkret implementiert wurden.

Kapitel 6 Die implementierten Änderungen werden hinsichtlich ihrer WebDAV Konformität geprüft und die Geschwindigkeit wird mit der Urspungsversion verglichen.

Kapitel 7 Abschließend wird das Ergebnis dieser Arbeit zusammengefasst und Weiterentwicklungsmöglichkeiten werden aufgezeigt.

2 Grundlagen & Hintergrund

2.1 Historischer Hintergrund Apache Webserver

Der Apache Webserver entstand im Jahr 1995. Apache basierte auf dem NCSA Server, der von National Center for Supercomputing Applications entwickelt wurde. Der Apache 1.0 erschien im Dezember 1995. Der Apache 1.x Server wurde kontinuierlich weiterentwickelt, bis zur Version 1.3, die im Juni 1998 erschien.

Im Jahr 2000 begannen die Apache-Entwickler die Einschränkungen der alten Architektur mit einer Neuentwicklung zu ändern. Die Apache 2.0 Freigabe erfolgte im April 2002, welche mehrere Verbesserungen gegenüber dem 1.x Zweig brachte.

- Verbesserte API
- Betriebsystemabstraktionsschicht
- Neue Extension Architektur

2.2 Apache 2.x Architektur

Der Apache HTTP Server 2.x wurde komplett neu geschrieben. Dabei wurde insbesondere darauf geachtet dass Plattformabhängigkeiten in die Apache Portable Runtime zu verlagern. Eine Besonderheit der Apache 2.x Architektur ist, dass die Multiprocessing Handhabung in auswechselbare Multi-Processing-Module verlagert wurden. Im Gegensatz zum Apache HTTP Server 1.x, der standardmäßig nur mittels Fork zur Parallelen Verarbeitung von Anfragen in der Lage ist. Dadurch ist die Version $>= 2.0$ in der Lage, das für das jeweilige Betriebssystem beste Modell auszuwählen.

2.2.1 Apache Portable Runtime

Die Apache Portable Runtime Bibliothek (APR) ist ein Produkt der Apache Software Foundation. Sie wurde entwickelt um plattformspezifische Unterschiede zu abstrahieren. Entwickelt wird die Bibliothek hauptsächlich für den Apache HTTP Server, was aber nicht bedeutet, dass andere Projekte die Bibliothek verwenden können.

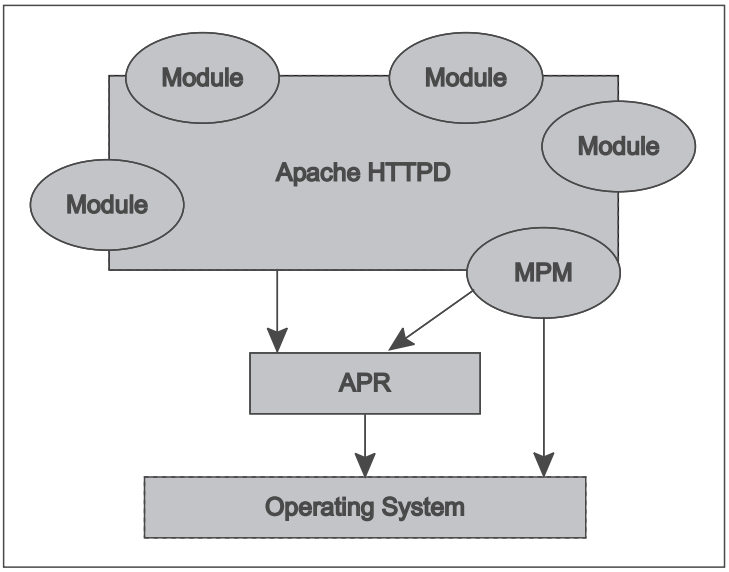

Abbildung 2.1: Apache 2.x Architektur Schema (Abbildung entnommen aus [Kew07])

Das populärste Beispiel für ein anderes Projekt, das die Runtime verwendet, neben Apache HTTP Server, ist das Versionskontrollsystem Subversion.

Die Apache Portable Runtime teilt sich in drei Unterprojekte auf.

APR Die Core Bibliotek mit allen Funktionen die üblicherweise benötigt werden.

APR-Util Erweitert die APR Bibliothek um weitere nicht immer benötigte Funktionen.

APR-Iconv Plattformunabhängige Implementation der iconv Bibliothek, benötigt für die Windows Plattform.

Ein elementaler Bestandsteil von Apache Portable Runtime sind die sogenannten *Pools*. Sie sind die Basis des Resourcen Managment in APR und dem Apache Webserver. Programmierer können statt mit der Standard C Funktion `malloc()` auch über *Pools* Speicher anfordern. Das hat den entscheidenden Vorteil das die APR sicherstellt, das der Speicher wieder freigebeben wird wenn er nicht mehr benötigt wird. Das wird erreicht indem es mehrere *Pools* gibt die sich in ihrer Lebenzeit unterscheiden. Der Programmierer muss somit nur noch entscheiden welche Lebenzeit der angeforderte Speicher benötigt.

2.2.2 Apache WebDAV Module - mod_dav

Das Apache Module mod_dav ist ein Standardmodul des Apache 2.x Webserver. Es erweitert den Apache Webserver um WebDAV Protokoll wie er in RFC2518[Weba] spe-

zifiziert wird.

Die Architektur von mod_dav ist in zwei Komponenten aufgeteilt. Zum einen das Modul mod_dav selbst welches das protokollspezifische Verhalten implementiert. Zur Speicherung der Daten und Metadaten definiert es apachetypisch «Hooks» auf denen die mod_dav Reposistory Module einhängen können. Die Repository Module implementieren die Art der Speicherung der Daten und den Zugriff auf diese Daten.

Neben den Hooks die für eine Implementierung des RFC2518 ausreichen, werden auch Hooks für die Unterstandards DASL[DAS] und DeltaV[Del].

Innerhalb einer Diplomarbeit[Lit06] im DLR wurde mod_dav um ACP[ACP] erweitert, ist aber noch nicht in der offiziellen Quellcode Distribution eingeflossen.

2.2.3 Catacomb - mod_dav_repos

Mod_dav_repos oder auch Catacomb genannt ist ein Repository Modul für mod_dav. Es kann alternativ zum standardmäßig ausgelieferten mod_dav_fs verwendet werden. Im Gegensatz zu diesem implementiert es neben dem Standardumfang des WebDAV Protokolls auch die Standards für Versionierung (DeltaV) und Serverseitige Suche (DASL). In einer Entwicklerversion existiert auch eine Version die das *Access Control Protocol* implementiert.

Catacomb verwendet anstatt des Dateisystems zur Speicherung der Dateien und Metadaten eine Relationale Datenbank. Optional kann mittlerweile ein Schwellenwert für die Dateigröße angegeben werden, ab welchem eine Datei auf dem Dateisystem abgelegt werden soll. Die Metadaten werden weiterhin in der Datenbank verwaltet.

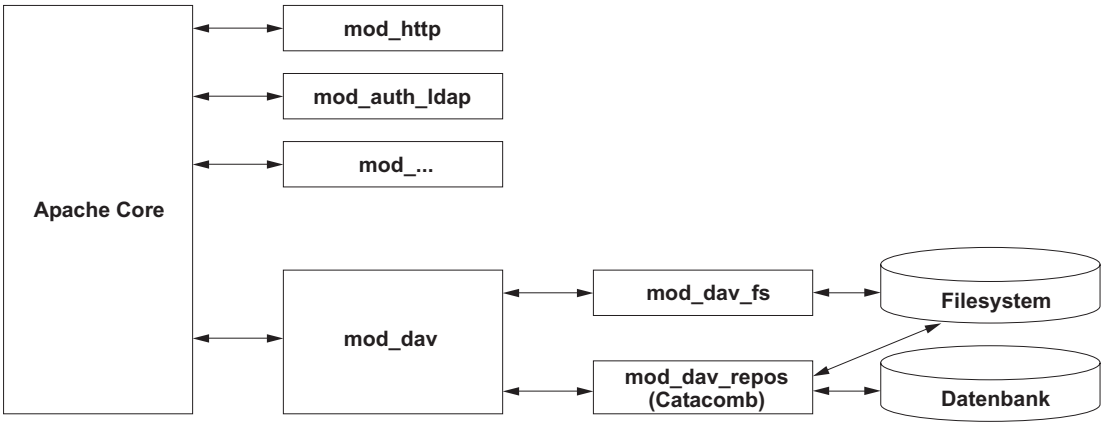

Abbildung 2.2: Schematischer Aufbau Catacomb

2.2.4 Apache DBD API

Der Ursprung der DBD API liegt bei dem Anwendungsentwickler Nick Kew. Er sah die Notwendigkeit einer Abstraktionsschicht für Datenbanken in der Apache Portable Runtime.

Bevor die Datenbank API apr_dbd in Apache Portable Runtime existierte, mussten Apache Module auf Datenbanken direkt über Datenbank spezifische API zugreifen. Daraus resultierten einige Module, die die gleiche Funktionalität lediglich mit unterschiedlichen Datenbanken implementierten. Dies ist eine unötige und fehlerträchtige Mehrfachentwicklung. Die Datenbankabstraktions API apr_dbd kann hier Abhilfe schaffen. Es wird nun nur noch die Funktionalität mit Hilfe der apr_dbd implementiert und hat somit automatisch alle von dbd unterstützten Datenbanken abgedeckt.

Als Vorlage für den Entwurf verwendete Kew die Datenbankabstraktionsschicht aus Perl, DBI/DBD. Daraus ergibt sich auch die Klassifikation der Abstraktionschicht, sie abstrahiert lediglich die unterschiedlichen APIs der Datenbanken auf eine einheitliche Schnittstelle.

Wie alle Module aus APR bzw. httpd verwendet es zu Speicherverwaltung Pools.

Die Datenbankabstraktionschicht ist seit Version 1.2 der Apache Portable Runtime bestandteil der APR-Util .

Architektur

Die Architektur gliedert sich in 4 Schichten für Apache-Module. Abbildung 2.3 zeigt den schematischen Aufbau dieser Schichten.

Die unterste Schicht stellen die DBD Treiber dar. Sie implementieren unter Verwendung der Datenbankspezifischen API, die apr_dbd API. Neben dem eigentlichem Ausführen der Statements, bedeutet das Reservieren und Zuweisen von Speicher aus dem Pool und SQL Statement Platzhalter durch datenbankspezifische zu ersetzen.

Apr_dbd stellt die zweite Schicht dar, es stellt den Apache Modulen oder der Programmen die die Runtime verwenden die Schnittstelle bereit. Es reicht im Wesentlichen die Methodenaufrufe zum Treiber durch.

Die Dritte Schicht ist mod_dbd, es ist ein Datenbankverbindungsmanagement Modul für apr_dbd. Bei Verwendung einer Thread Plattform (MPM) verwaltet es einen Pool von dauerhaften Datenverbindungen. Bei nicht Thread basierenden Plattformen verwaltet es eine einzelne Datenbankverbindung pro Prozess. Anwendungsmodule müssen sich somit nicht um den Verbindungsaufbau bzw. -abbau kümmern, sofern sie mit einer Datenbankverbindug pro Request auskommen.

Schicht Vier stellen die Apache Module dar, die dbd verwenden.

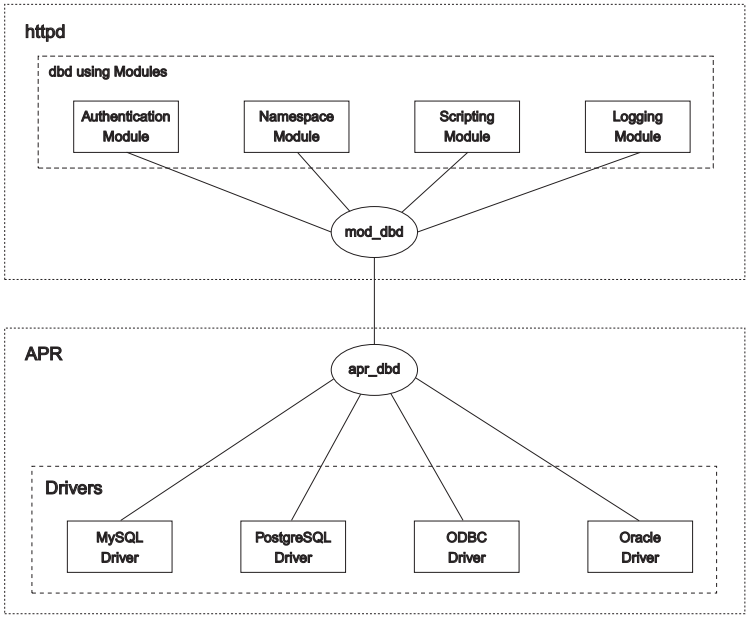

Abbildung 2.3: Architektur DBD (basierend auf einer Abbildung aus [Kew07])

Funktionsumfang & Treiber

In der Version 1.2 unterstützt apr_dbd nur eine sehr einfache Abstraktion, die API kennt nur Strings/VARCHAR als Datentypen. Die API lässt sich in sieben Kategorien einteilen.

- SQL Statements vorbereiten (prepare)
- SQL Statements ausführen ohne Ergebnis (query)
- SQL Statements ausführen mit Ergebnismenge (select)
- Operation auf die Ergebnismenge (z.B. get_row)
- SQL Transaktionen
- Sonstiges, Escape strings, Error Handling

2.2.5 AAA - Access, Authentication and Authorization

Schon seit den Versionen 1.x und 2.0 existiert ein vorgeschalteter Mechanismus für Apache, der die Zugriffskontrolle, Authentifiktion und Autorisation übernimmt. Mit der Version 2.1/2.2 wurde dieser Bereich komplett überarbeitet und in vier Teile/Module (siehe Abbildung 2.4) unterteilt.

Access Hier kann anhand z.B. Hostname, IP-Adresse oder MAC-Adressen gefiltert werden.

Auth HTTP bietet zwei Arten sich zu authentifizieren Basic und Digest. Basic ist eine in Base64 kodierte Kombination aus Benutzername und Passwort. Das Standardmodul mod_auth_basic erledigt die Dekodierung, delegiert die Verifikation der Kombination and *authn*. Bei Digest wird statt des Passworts der MD5-Hash des Passworts geschickt.

Authn Verifiziert User/Passwort oder User/Passworthash

Authz Erlaubt das Authentifizierten, gewährt oder verweigert den Zutritt.

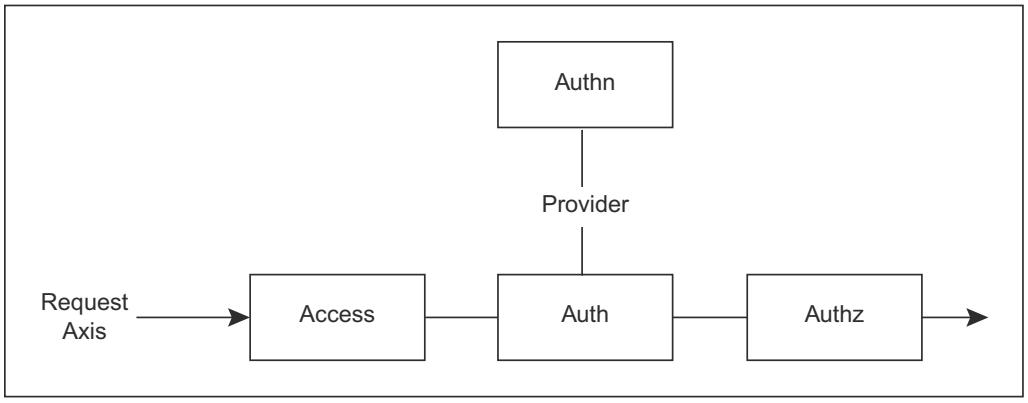

Abbildung 2.4: Access Control, Authentication and Authorization

2.3 WebDAV

WebDAV besteht aus einer Reihe von Erweiterungen für das Hypertext Transfer Protokoll (HTTP) in der Version 1.1, welches den Benutzern das gemeinsame Bearbeiten und Verwalten von Dateien auf dem Server erlaubt.

Neben den in HTTP definierten HTTP-Methoden, die nur lesenden Zugriff bieten, wurden neue Methoden eingeführt, die dem Beutzer auch lesenden Zugriff bieten.

Durch seine Verwandschaft zu HTTP können dieselben Firewall Regeln Verwendung finden können wie bei HTTP. D.h. es sind keine zusätzlichen potenziellen Sicherheitsgefährdenden Löcher in die Firewall zu machen.

2.3.1 Historischer Hintergrund

Historisch betrachtet beginnt WebDAV schon mit HTTP, dass Tim Berners-Lee am CERN entwickelte. Berners-Lee hatte von Anfang an beabsichtigt sowohl das Schreiben als auch Lesen von Webseiten zu ermöglichen. Die meisten Webserver implementierten nur den lesenden Zugriff.

1996 formierte sich eine Arbeitsgruppe von Unternehmen und Organisationen um auf Basis von HTTP einen neuen Standard für verteiltes Schreiben und Versionierung (Distributed Authoring and Versioning) von Webinhalten zu entwickeln.

Aufgrund der Komplexität von verteiltem Schreiben, entschied man sich WebDAV ohne die Versionierung zu standardisieren. Dafür wurden für die Versionierung (DeltaV) wie auch für die Suche und Lokalisierung (DASL) spezialisierte Arbeitsgruppen eingerichtet.

1996	Gründung der WebDAV Working Group
1998	Entscheidung den Standard ohne Versionierung und Suche weiterzuentwickeln. Einrichtung spezialiserter Working Groups für Versionierung (DeltaV) und Suche (DASL)
1999	Veröffentlichung des «HTTP Extensions for Web Distributed Authoring and Versioning» RFC2518
2002	Veröffentlichung des «Versioning Extensions to WebDAV» RFC3253, Wiederaufnahme der Arbeit an DASL
2004	Veröffentlichung des «WebDAV Access Control Protocol» RFC3744
2007	WebDAV Working Group beendet seine Arbeit am Standard nach einem inkrementellen Update des RFC2518 auf RFC4918

Tabelle 2.1: Zeitlicher Ablauf der WebDAV Protokoll Entwicklung

2.3.2 HTTP Protokollerweiterung

WebDAV baut auf HTTP 1.1 auf und verwendet deshalb die acht bereits definierten Methoden weiter. Die Semantik der Methoden bleibt weitestgehen erhalten.

- GET
- POST
- OPTIONS
- PUT

- DELETE
- HEAD
- TRACE
- HEAD

Zusätzlich werden in der Grundversion von WebDAV [Webb] sechs weitere Methoden eingeführt.

- PROPPATCH
- PROPFIND
- MKCOL
- MOVE
- LOCK
- UNLOCK

Nicht jeder WebDAV Server muss den vollen Funktionsumfang implementieren. Zum Bekanntgeben welcher Funktionsumfang vom Server unterstützt wird, definiert WebDAV drei Klassen. In der Klasse 1 muss alles aus dem RFC4918[Webb] ohne die LOCK Funktionalität implementiert sein. Klasse 2 setzt automatisch das erfüllen der Klasse 1 voraus plus LOCK Funktionalität. Bei Klasse 3 muss explizit das RFC2518[Weba] implementiert werden. Auch bei Klasse 3 muss Klasse 1 unterstützt werden.

2.3.3 Metadaten

In WebDAV unterscheidet man zwischen Daten und Container. Container fassen Daten und weitere Container zusammen und erlauben somit eine hierarchische Strukturierung der Daten. Daten werden in WebDAV Resourcen und Container als Collections bezeichnet. Als Analogie zu Dateisystemen kann am Resourcen mit Dateien und Collections mit Verzeichnissen vergleichen. Es gibt Server Implementationen die diese Zuweisung eins zu eins verwenden.

Neben der Verwaltung von Resourcen und Collections kann WebDAV zu jeder Resource Metadaten speichern. Als Metadaten bezeichnet man Daten die zusätzlichen Informationen über die Resource bereitstellen. Sie sind aber nicht Teil der eigentlichen Resourceninhalts. Beispiele für Typische Metadaten oder im WebDAV Kontext Properties genannt, sind z.B. der Autor, Sprache oder Größe eines Dokuments.

In WebDAV muss man zwischen zwei Arten von Metadaten unterschieden werden.

Live Property Hier muss zwischen zwei Arten unterschieden werden. Metadaten deren Inhalt vom Server verwaltet werden, die Größe (contentlength) einer Resource zum Beispiel. Und Metadaten die vom Client erzeugt werden, hier prüft der Server auf Einhaltung der Syntax. Die Sprache (contentlanguage) ist ein Beispiel dafür.

Dead Property Sind Metadaten die vom Benutzer beliebig erzeugt, verändert und gelöscht werden können. Der Server prüft nicht Syntax des Inhalts.

2.3.4 DASL

Das DAV Searching & Locating erweitert WebDAV um serverseitige Suche. DASL erlaubt das Suchen in dem Inhalt von Resourcen und deren Properties. Bei der serverseitigen Suche wird die Suchanfrage in XML dem Server übermittelt. Im Gegensatz zu einer clientseitigen Suche muss nicht erst die gesamte Resourcen Hierarchie geladen werden, um Suchanfragen zu verarbeiten zu können. Zudem wird erst durch das serverseitige Suchen, das Suchen in Resourceninhalten praktikabel, denn das gesamte Herunterladen der Daten beeinträchtigt zu einem die Serverlast und erhöht die Komplexität für den Client erheblich.

DASL spezifiziert eine neue HTTP Methode SEARCH und eine XML Grammatik für Suchanfragen. Diese Grammatik (basicsearch) ist in ihrem Aufbau einer einfachen SQL Abfrage sehr ähnlich. Genauso wie SQL besteht es aus zwei bis fünf Teilen und verwendet die selben Schlüsselwörter.

select wählt die Properties aus, die in der Ergebnismenge zurückgeliefert werden sollen.

from definiert den Bereich auf den sich die Suchanfrage bezieht. Eingegrenzt wird der Bereich durch Angabe einer URI zusammen mit der Tiefe. Optional kann auch anhand der versionsnummer der Bereich eingegrenzt werden.

where definiert die Bedingungen anhand derer entschieden wird ob die Resource Bestandteil der Ergebnismenge ist oder nicht. Bedingungen können für Properties und für den Inhalt der Resource erstellt werden.

orderby legt die Sortierung fest in der die Ergebnismenge zurückgeliefert werden soll.

limit begrenzt die Ergebnismenge.

Die DASL Arbeitsgruppe hat noch keine fertige Version veröffentlicht und befindet sich somit noch im Draft Status. Den aktuellen Entwurf finden Sie unter [DAS].

Feature / Package	Core-Versioning	Basic-Server-Workspace	Basic-Client-Workspace	Advanced-Server-Workspace	Advanced-Client-Workspace
Version-Control	x	x	x	x	x
Checkout-In-Place		x		x	
Label			x		x
Update			x		x
Workspace		x		x	
Working-Resource			x		x
Version-History		x		x	
Merge				x	x
Baseline				x	x
Activity				x	x
Version-Controlled-Collection				x	x

Tabelle 2.2: DeltaV Features

2.3.5 DeltaV

DeltaV erweitert WebDAV um Versionierung. Es stellt die wichtigste Erweiterung des WebDAV Standards dar, damit erhält das V im Namen seine Berechtigung. DeltaV, das in RFC3253[Del] beschrieben wird, spezifizert eine Reihe von Features, die nachfolgend noch genauer erläutert werden. Dabei muss ein Server der Versionierung unterstützt muss mindestens das Feature *Version-Control* implementieren.

Um die Server und Client Entwicklung zu erleichtern, werden sogenannte Packages spezifiziert. Packages definieren sinnvolle Kombinationen von Features. Das *Core-Versioning* Package stellt das kleinste Featureset dar, das unterstützt werden kann. In der Tabelle 2.2 werden alle Packages und ihre benötigten Features dargestellt.

In der folgenden Aufstellung werden die im Standard definierten Features kurz erläutert. Für genaue Informationen verweise ich auf [Del] und [Dus04].

Basis Versioning Features:

Version-Control definiert wie Resourcen unter Versionskontrolle gestellt werden können. Dieses Feature führt die zwei Methoden VERSION-CONTROL und REPORT ein.

Checkout-In-Place definiert checkout, checkin und uncheckout Funktionalität. Mit diesem Feature werden auf die Methoden CHECKIN, CHECKOUT und UNCHECKOUT eingeführt.

Label definiert wie Versionen mit einem Label zu versehen können. Die Methode LABEL wird hiermit eingeführt.

Update definiert das Ändern des Inhalts und Properties eingecheckter Ressourcen auf eine bestimmte Version. Zu diesem Zweck wird die Methode UPDATE eingeführt.

Workspace definiert einen privaten Arbeitsbereich für Benutzer. Hiermit wird die Methode MKWORKSPACE eingeführt.

Working-Resource definiert Working Copies einzelner Resourcen, die privat bearbeitet werden. Dieses Feature führt keine weiteren Methoden ein, erweitert lediglich die Semantik der vorhandenen Methoden.

Version-History definiert den Zugriff auf vorherige Versionen einer Resource. Wie bei Feature Working-Resource führt Version-History keine weiteren Methoden ein und erweitert lediglich die Semantik der bereits vorhandenen Methoden.

Erweiterte Versionierungs Features:

Merge definiert das Zusammenführen (merge) zweier Resourcen. Die Methode MERGE wird hiermit eingeführt.

Baseline definiert, wie eine Baseline definiert werden kann, also einen Zustand mehrerer Resourcen mit einem Tag kennzeichnen. Die Methode BASELINE-CONTROL wird hiermit eingeführt.

Activity definiert, wie eine logische Änderung, die mehrere Resourcen betrifft durchgeführt werden. Die Methode MKACTIVITY wird hiermit eingeführt.

Version-Controlled-Collection definiert, wie Änderungen von Collection verfolgt werden. Dieses Feature führt keine weiteren Methoden ein, erweitert lediglich die Semantik der vorhandenen Methoden.

2.3.6 Lock

Beim zeitgleichen Arbeiten am selben Dokument wird es unvermeidlich zu dem Problem kommen, dass sich die Änderungen überschreiben. Dieses Problem ist unter dem Namen Lost-Update bekannt. Eine wirkungsvolle Methode diesem Problem zu begegnen ist die Einführung von Locks.

WebDAV definiert zwei Arten von Locks:

Shared Lock Bei Shared Locks können beliebig viele Locks von verschiedenen Benutzern erzeugt werden. Sollte beim Erzeugen eines Shared Locks auf eine Ressource schon ein anderer Shared Lock existieren, wird der Benutzer darauf hingewiesen. Möchte er trotzdem eine Schreiboperation durchführen, wird er nicht darangehindert. Um zu verhindern, dass eigene Änderungen oder die eines anderen Benutzer nicht unbeabsichtigt überschrieben werden, müssen von den Benutzern Absprachen getroffen werden.

Exclusive Lock Bei Exclusive Locks kann pro Ressource nur ein Lock zur selben Zeit vergeben werden. Es darf nur der Eigner des Lock Tokens Änderungen an der Ressource durchführen, bis der Lock wieder freigegeben wird.

Für beide Arten von Locks gilt, dass der Client für ein Lock eine Lebensdauer (Timeout) anfordert. Diese Lebensdauer kann von einigen Sekunden bis unendlich lang sein. Wobei der Server kann die angeforderte Lebendauer ändern kann. Wird vor Erreichen dieser Grenze der Lock nicht zurückgesetzt, löscht der Server selbstständig nach dem ablaufen der Lebensdauer den Lock. Dies hat den Sinn das Lock von Clients die vergessen haben Locks zu löschen oder abgestürzt sind nicht im System verwaisen.

2.3.7 ACP

Mit Hilfe des Access Control Protokoll [ACP] können Zugriffsrechte für Benutzer (Principals) bestimmt werden. Dabei können einem explizit Rechte verliehen (grant) oder verweigert (deny) werden.

Zugriffsrechte auf eine Resource werden in Access Control List (ACL) verwaltet. Jede ACL ist einer *Prinicipal* oder einer Gruppe zugeordnet. Eine ACL besteht wiederum aus Access Control Entries (ACE) die jeweils Rechte erlauben oder verweigern.

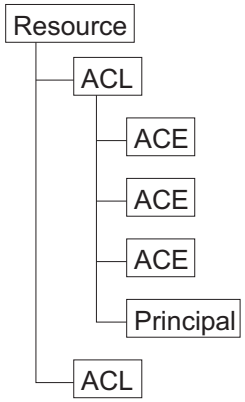

Abbildung 2.5: Schema ACL

Jahr	Name	Kommentar
1986	SQL1 bzw. SQL86	Erstmalige Standardisierung SQL durch ANSI. 1987 von ISO übernommen.
1989	SQL89	Kleinere Änderungen.
1992	SQL2 bzw. SQL92	Erste größer Überarbeitung des Standards.
1999	SQL3 bzw. SQL99	Unter anderem Einführung objektorientierten Features.
2003	SQL:2003	Einführung XML spezifischer Features.
2006	SQL:2006	Starke Erweiterung der XML Features.

Tabelle 2.3: Historischer Abriss SQL Standards

Neue Resourcen erben beim Erstellen die ACL der übergeordneten Collection.

Zum Ändern der Rechte wird eine weitere HTTP Methode «ACL» eingeführt.

2.4 Datenbanken

2.4.1 Relationale Datenbanken

Relationale Datenbanken basieren auf dem Relationenmodell von Edgar F. Codd. Die Grundlagen der Theorie dieses Modells erstellte Codd in den 60ern und 70ern. Ihre Grundlage dieses Modells stellt die Relation dar, die man bildhaft sich als Tabelle vorstellen kann. In einem relationalen Datenbank können beliebig viele Tabellen existieren, die über Operationen der relationalen Algebra miteinander verknüpft werden. Die relationale Algebra besteht aus den Grundoperationen Selektion, Vereinigung, Umbenennung, Projektion, Differenz und Kreuzprodukt.

2.4.2 SQL

SQL steht für Structured Query Language manchmal auch für Standard Query Language. Es ist eine von der ISO standardisierte Abfragesprache für Relationale Datenbanken. Der theoretische Hintergrund der Sprache ist die relational Algebra nach Codd.

Die Tabelle 2.3 zeigt den chronologische Abfolge der verabschiedeten SQL Standards.

Mit dem Standard SQL-92 oder auch SQL-2 genannt wurden die Statements in die drei Kategorien DML, DDL und DCL eingeteilt.

DML Data Manipulation Language - SELECT, INSERT, UPDATE, DELETE

Klasse	Beschreibung	Beispiel Kommandos
SQL Connection Statements	starten und beenden einer Client Verbindung	CONNECT, DISCONNECT
SQL Control Statements	Steuerung der Ausführung einer Reihe von Statements	CALL, RETURN
SQL Data Statements	Operationen auf Relationen einer Datenbank	SELECT, INSERT, UPDATE, DELETE
SQL Diagnostic Statements	Diagnose Informationen und Ausnahmen und Fehler hervorrufen	GET DIAGNOSTICS
SQL Schema Statements	Operationen auf Schemas und Objekte einer Datenbank	CREATE, ALTER, DROP
SQL Session Statements	Verändern des Standardverhaltens und andere Parameter einer Sitzung	SET
SQL Transaction Statements	Setzen des Beginns und Ende einer Transaktion	COMMIT, ROLLBACK

Tabelle 2.4: SQL99 Statement Klassifikation

DDL Data Definition Language - CREATE, ALTER, DROP

DCL Data Control Language - GRANT, REVOKE

Mit dem Standard SQL-99 wurde eine erweiterte Klassifikation der Statements eingeführt, Tabelle 2.4 zeigt die neue Einteilung in sieben Kategorien. Häufig wird aber noch die alte Einteilung (DDL, DML, DCL) verwendet.

Datentypen

In SQL99, wie auch in den Standards zuvor werden eine Reihe von Datentypen definiert, die jede Datenbank die SQL99 implementiert unterstützen sollte. Tabelle 2.5 zeigt eine Aufstellung der definierten Datentypen.

User Defined Funktionen (UDF)

SQL99 definiert eine Möglichkeit dem Benutzer eigene Funktionen in seiner Datenbank zu erstellen. User-defined Functions können genauso wie systemeigene Funktionen in SQL Statements verwendet werden.

Kategorie	Datentype SQL99
binary	binary large object (BLOB)
bit string	bit bit varying
numeric	integer (INT) smallint numeric decimal float real double precision
character	char varying char (VARCHAR) national caracter (NCHAR) national character varying (NVARCHAR) character large object (CLOB) national character large object (NCLOB)
boolean	boolean
datetime	date time time with timezone timestamp timestamp with timezone interval

Tabelle 2.5: SQL99 Datentypen

2.4.3 Transaktionen

In der Informatik bezeichnet man eine Transaktion als eine Folge von Operationen die als logische Einheit zu betrachten ist. Am häufigsten werden Transaktionen im Kontext von Datenbanken benutzt.

Transaktionen haben im Kontext von Datenbanken zwei Aspekte zu erfüllen. Zum einen muss Sie den konkurrierenden Zugriff auf die Datenbank handhaben, zum Anderen im Fehlerfall eine konsistente Datenbank hinterlassen.

ACID

Zum Zweck einer korrekten Synchronisation und korrektem Fehlerverarbeitung sind die folgenden vier Eigenschaften [Vos00] zu erfüllen. Sie werden auch ACID-Prinzip genannt.

Atomicity (Atomarität) Die Befehlsfolge wird entweder vollständig ausgeführt oder gar nicht.

Consistency (Konsistenz) Die Integritätsbedingungen der Datenbank werden eingehalten, d.h. eine Transaktion hinterlässt die Datenbank in einem «konsistenten» Zustand.

Isolation (Isolation) Transaktionen laufen isoliert von anderen Transaktionen, d.h. die Transaktion sieht nur «gesicherte» Daten aus der Datenbank.

Durability (Persistenz) Änderungen einer erfolgreich beendeten Transaktion müssen dauerhaft sein.

Wenn das Transaktionsverwaltungssystem eines Datenbankmanagementsystems diese vier Eigenschaften gewährleistet.

Kann eine Transaktion nicht erfolgreich zu Ende gebracht werden, so ist gewährleistet:

1. die Transaktion hinterlässt keine Spuren . (*atomicity*)
2. sich die Datenbank in einem Konsitenten Zustand vor der fehlerhaften Transaktion befindet. (*durabilitiy* und *atomicity*)
3. im Mehrbenutzerbetrieb keine «Anomalien» auftreten. (*isolation*)

2.5 BASE64 Codierung

Base64[bas] ist ein Kodierverfahren, welches zur Kodierung von Binärdaten eine Zeichenfolge aus Codepageunabhängige ASCII-Zeichen benutzt.

Zur Kodierung werden die Zeichen A-Z, a-z, 0-9, + und /, verwendet. Als Ende Zeichen dient das =. Bei Verwendung dieser Kodierung steigt der Platzbedarf um 33% gegenüber dem Orginal.

3 Datenbankmodell

WebDAV ist wegen seiner Verschachtelung von Collections eine hierarchische Struktur. Für das Standard mod_dav Repository Modul mod_dav_fs dass das Dateisystem als Speicherort verwendet, ist es eine eins zu eins Zuordnung von Resourcen und Collections auf Dateien und Verzeichnisse. Die Hierarchie ist implizit durch die Schachtelung der Verzeichnisse vorgegeben.

Catacomb verwendet zur Datenhaltung eine Relationale Datenbank. Um hierarchische Daten in relationalen Datenbanken abbilden zu können, muss die Hierarchieinformation explizit in einem Feld der Tabelle gespeichert werden. Bei Catacomb ist es das Feld URI in der Tabelle dasl_resource, es enthält in einem String den Pfad zu der Resource oder Collection.

Den Inhalt der einzelnen Tabellen aus Abbildung 3.1 wird nachfolgend kurz erläutert.

dasl_resource Enthält die Resourcen, Collections und die dazugehörigen Live Properties.

dasl_property Enthält die Dead Properties für Resourcen und Collections, verknüpft mit der Tabelle dasl_resource über die Spalte serialno.

dasl_namespace Enthält alle Namespaces der Properties zur Vermeidung von Namenskonflikten.

dasl_lock Enthält Lockinformationen der aktuell vergeben Locks. Verknüpft mit dasl_resource über serialno.

dasl_locknull Enthält Lockinformationen für Nulllocks, Nulllocks können erstellt werden zum reservieren eines Namens für eine Resource oder Collection.

version_resource Enhält von Versionierten Resourcen die alten Versionen mit einer Versionsnummer versehen.

version_property Enhält von Versionierten Resourcen deren alte Properties mit einer Versionsnummer versehen.

Für die ACP Protokollerweiterung musste das Datenbankmodell um weitere Tabellen erweitert werden. Der Tabelle dasl_resource kommt eine besondere Bedeutung zu. Sie enthält dann zusätzlich zu den Resourcen und Collection die Benutzer und Gruppen also die Principals.

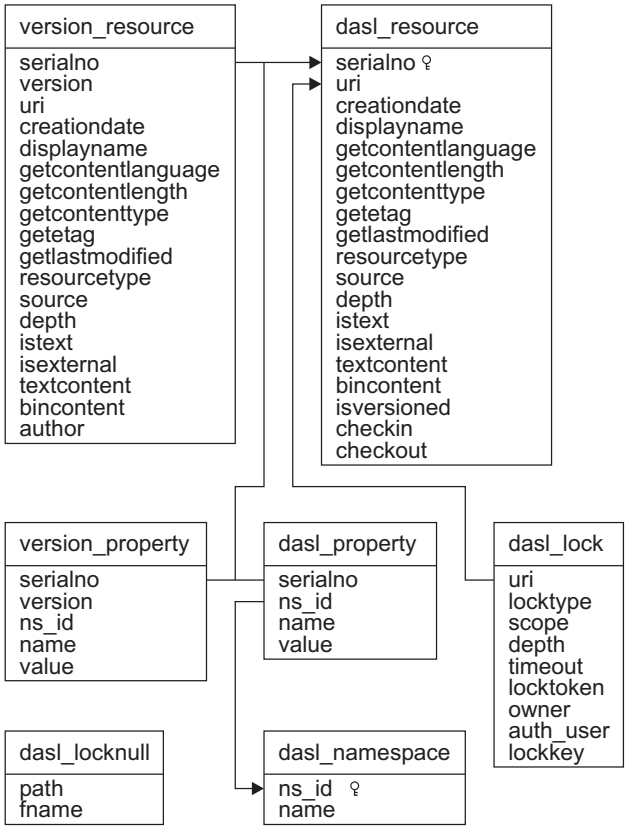

Abbildung 3.1: Datenbankmodell

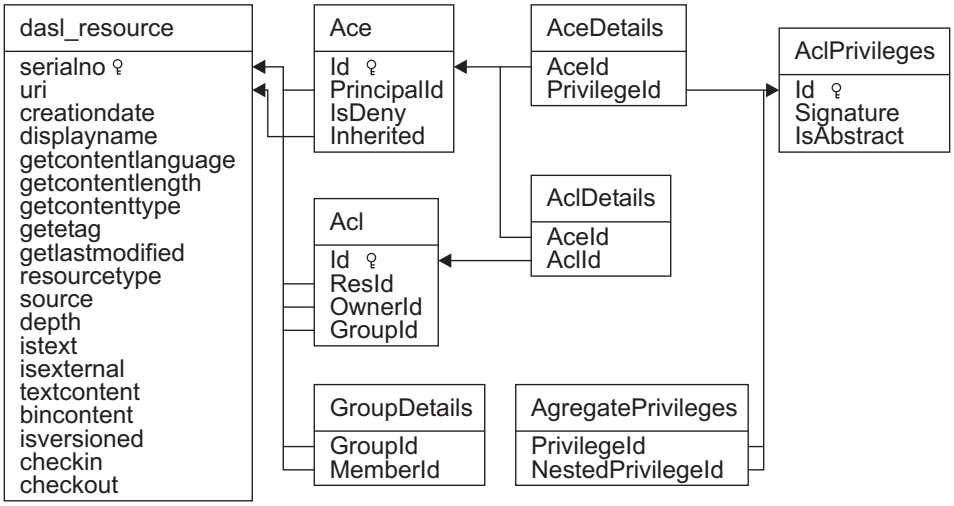

Abbildung 3.2: Datenbankmodell ACL

Die Tabellen Acl, AclDetails, Ace und AceDetails sind für die Zuordnung der Resource zu ihren Principals und Privileges zuständig. Die Tabellen AclPrivileges definiert welche Privileges existieren. Durch die Tabelle AgregatePrivileges können auch zusammengesetzte Privileges existieren. Die Tabelle GroupDetails erledigt die Zuordnung der Benutzer zu den Gruppen. Die Abbildung 3.2 zeigt die Beziehungen zwischen den Tabellen.

4 Analyse und Entwurf

4.1 Datenbankabstraktion

4.1.1 Allgemein

Der Catacomb in der bisherigen Version definiert eine eigene API die von Datenbank-spezifischen API abstrahiert. Theoretisch kann durch eine Neuimplementation der, in `dbms_mysql.h` deklarierten, Methoden Signaturen Catacomb auf andere Relationale Datenbanken portiert werden. Nach Analyse des Quellcodes ergeben sich praktisch ein paar Schwierigkeiten.

Zum einen existieren im eigentlich datenbankunabhängigen Teil SQL Statements die sehr MySQL zentrisch sind. Vorallem die Statements für das Einlesen und Auslesen der Ressourcen sind unter dem Gesichtpunkt der datenbankunabhängigkeit problematisch. Der SQL Befehl

```
1 SELECT textcontent
2 INTO DUMPFILE ?
3 FROM dasl_resource
4 WHERE serialno=?
```

und die MySQL spezifische SQL-Funktion `LOAD_FILE(?)` die zum auslesen bzw. einlesen verwendet werden, haben zudem das Problem, dass der Webserver und der Datenbankserver auf dasselbe Dateisystem zugreifen können müssen. Außerdem muss in der jetzigen Implementation die gleiche Verzeichnishierarchie gewährleistet sein, da in beiden Fällen als Parameter eine absolute Pfadangabe erwartet wird.

Zum anderen wird an einigen stellen die MySQL API und Datenstrukturen ohne über die eigene Abstraktionsschicht zu gehen. Hier ist insbesondere die MySQL C Funktion `mysql_insert_id()` zu nennen.

Die Catacomb eigene Abstraktionsschicht dbms ist von seinem groben Ablauf apr_dbd sehr ähnlich. Die Folgenden zwei Aufzählungen zeigen den groben Ablauf beider APIs, wobei jeder punkt ein Funktionsaufruf darstellt. Die erste Aufzählung zeigt dbms die zweite apr_dbms.

1. Prepare SQL Statement
2. Parameter 1 - n setzen
3. Statements ausführen

4. Zeile(n) holen
5. Statement explizit schließen

1. Prepare SQL Statement
2. Parameter setzen und ausführen
3. Zeile(n) holen, Statement wird implizit durch ereichen der letzten Zeile geschlossen

Der größte Unterschied ist das implizite Schließen des Statements nach Erreichen der letzten Zeile. Dadurch muss durch die gesamte Ergebnismenge iteriert werden, obwohl vielleicht nur die erste Zeile benötigt wird.

Aufgrund der großen Ähnlichkeit ergeben sich zwei Möglichkeiten apr_dbd einzuführen. Die Erste ist mit apr_dbd einen neuen dbms Treiber zu implementieren und die API erweitern. Der Vorteil ist es muss insgesamt weniger Code angepasst werden. Nachteil ist dass eine zusätzliche Schicht eingefügt wird.

Die Zweite ist dbms komplett durch apr_dbd zu ersetzen. Der Vorteil ist, es wird keine zusätzliche Schicht eingefügt die zu zusätzlichen Performance einbußen führen kann. Der Nachteil ist, dass deutlich mehr Quellcode angepasst werden muss.

Unabhängig von beiden Möglichkeiten, müssen in `dbms.c` alle MySQL API und SQL Abhängigkeiten entfernt werden.

WebDAV Server haben keine speziellen Anforderungen an bestimmte Reaktionszeiten, aber es schadet nicht, schon beim Entwurf auf Geschwindigkeit zu achten. Zudem gewährleistet Apache Portable Runtime innerhalb Major Versionen dass die API aufwärts kompatibel bleibt. Unter diesen Voraussetzungen bietet es sich an die dbms API komplett durch apr_dbd zu ersetzen.

4.1.2 ACL

Die im ACL Teil verwendeten SQL Statements sind alle Standard konform. Somit beschränkt es sich hier auf das ersetzen der alten API durch apr_dbd und der direkten Verwendung der MySQL API.

4.1.3 Volltextsuche

Die Volltextsuche ist ein kritischer Punkt im Hinblick auf Datenbankabstraktion. Die Volltextsuche ist eine sehr von der Datenbank abhängige Funktion. Da gibt es drei Punkte, die bei der Überführung der MySQL Volltextsuche zu berücksichtigen sind.

1. Nicht jede Datenbank implementiert Volltextsuche
2. Es gibt keine einheitliche Syntax

3. Abhänigigkeiten innerhalb des Datenbanksystems, z.B. Tabellentyp MyISAM in MySQL oder PostgreSQL erfordert zusätzliche Tabellen und Spalten

Hauptsächlich wegen der nicht vorhanden einheitlichen Syntax für Volltextsuche, bietet es sich an, hier die Funktionalität in einer eigenen SQL Funktion zu verlagern. Durch die Funktion können Datenbank eigene Funktionen und Syntax verwendet werden ohne die Komplexität für der Volltextsuche zu erhöhen. Auch Datenbanken ohne Volltextsuche können verwendet werden, die Volltextsuche liefert in diesen Fällen immer nur den Relevanzwert 0.0.

4.1.4 apr_dbd bedingte Einschränkungen

Die Apache Portable Runtime in der Version 1.2.x hat im Bezug auf apr_dbd einen API bedingten Nachteil. Sie sieht momentan ausschließlich C-Strings als Parameter vor. Diese Einschränkung wirkt sich an aus, wenn Binäre Daten gespeichert werden sollen.

Das Problem besteht darin, dass in der Programmiersprache *C* Strings aus einem Array des Typs Char besteht, dessen Ende durch durch den Nummerischen Wert 0 gekennzeichet wird. Nun kann es vorkommen, dass innerhalb des zu speichernden Bytestroms der Wert 0 vorkommt. Auf Grund der Tatsache, dass die Null das Ende eines Strings markiert, würden nur die Daten bis zum auftreten der ersten Null gespeichert.

Eine vorübergehende Lösung bis die Runtime korrekt mit Binären Daten umgehen kann, ist den Bytestrom in Base64 kodiert in der Datenbank zuspeichern. Base64 bietet sich an, da es Bestandteil der Apache Portable Runtime ist. Der Nachteil ist der um 33% höhere Platzverbrauch der Resource im Vergleich zur unkodierten Bytestrom.

4.2 Transaktionen

Transaktionen haben ja nicht nur den Zweck Modifikationen an der Datenbank nur komplett durchzuführen. Sie haben auch bei Select Statements Sinn. Sie serialisieren auch den Zugriff auf den Datenbestand. Sie stellen damit sicher, dass ein Select Statement am Anfang einer Transaktion und dasselbe Statement am Ende der Transaktion zum selben Ergebnis kommen, obwohl in einer parallelen Transaktion die betreffende Tabelle verändert wurde. Deswegen bietet es sich an, dass Starten und Beenden der Transaktion möglichst früh stattfinden zulassen. Mit den in mod_dav definierten Hooks ist es leider nicht möglich, eine geeignete Stelle im Repository Modul zu finden, die mit vertretbaren Aufwand zu implementieren ist.

Die Einfachste Methode Transaktionen zu Implementieren ist, mod_dav um einen weiteren Hook zu erweitern. Dieser Hook besteht vorerst aus den beiden Funktionen *start* und *stop*.

4.3 Datenbank & SQL

Unterschiedliche Relationale Datenbanken unterstützen in der Regel nicht diesselbe Syntax, selbst die SQL Standards werden von den meisten nicht vollständig unterstützt oder weisen Herstellerspezifische Erweiterungen und Variationen auf. Wenn man sich auf die wesentlichen SQL Befehle aus DML beschränkt, kann dass zu einer hohen wahrscheinlichkeit jede Datenbank mit SQL Frontend verstehen.

4.3.1 Datenbankabhängigkeiten

Neben der Abstraktionsschicht muss auch darauf geachtet werden, dass Datenbankdesign nicht an spezielle Eigenheiten einer Datenbank anpasst wird. Jede Datenbank hat mehr oder weniger Eigenheiten die nicht auf jeder Datenbank laufen.

Ein Aspekt sind die selbst inkrementierende Ids der Tabellen, praktisch jedes Datenbanksystem implementiert selbst inkrementierende Ids anders. Neben dem Inkrementieren ist auch entscheidend wie man an die zuletzt eingefügte Id herankommt, ohne im Programmcode anhand des Datenbanktyps erst entscheiden zu müssen, welche Methode die richtige ist. Wie schon bei der Volltextsuche bietet sich eine UDF auf der Datenbank zu hinterlegen anhand dem Tabellennamen und des Spaltennamens den richtigen Wert zurückliefern.

```
1 LAST_INSERT_ID_BY_TABLENAME_AND_COLUMN(varchar,varchar )
```

4.3.2 Datentyp Matching

Anhand der schon vorhanden Tabellen Definitionen für die MySQL Datenbank, kann ein Überführung in die SQL-99 Datentypen durchführen. Anschließend können diese auf die Datenbankeigenen Datentypen überführt werden. Die meisten Datenbanken übersetzen automatisch SQL-99 Datentypen in eigene Datentypen. Aber trotzdem ist es notwendig jeweils eigene Tabellen Definitionen für jede Datenbank zu erstellen. Kleine Variationen in der `CREATE TABLE` Syntax erfordern das.

SQL-99	MySQL	PostGreSQL
smallint	smallint	int2
integer	int	int4
-	bigint	int8
blob	longblob	bytea
clob	longtext	text
varchar	varchar	varchar

Tabelle 4.1: Datentyp Matching SQL-99, MySQL und PostGreSQL

5 Implementierung

5.1 Programmierstil

Das Projekt Catacomb selbst definiert keinen Programmierstil. Aber es wird versucht anhand der schon existierenden Quellcodes Richtlinien zu extrahieren. Und die eigenen Progrogrammiergewohnheiten an den jeweils vorgegeben Stil anzupassen.

Prinzipiel würde es sich anbieten, sich an den Programmierrichtlinien des Apache Webservers zu orientieren. Es würde eine Integration in die offizielle Apache Webserver Distribution erleichtern.

5.2 Unit-Testing

Unit Tests sollten schon während der Entwurfsphase einer Software berücksichtigt werden. Umfangreiches Unit Testing bedeutet, erst einmal mehr Implementierungsaufwand aber erleichtert im Laufe der Zeit die Entwicklung erheblich.

Während der Implementierung wurden kein Unit Tests geschrieben. Leider existieren in Catacomb keine Unit Tests auf denen man aufbauen könnte. Deswegen hätte er grundlegende Infrastruktur entworfen und implementiert werden müssen. Bei der Infrastruktur müssten diverse, nicht triviale, Abhängigkeiten zu einer Datenbank und dem Apache/ mod_dav berücksichtigt werden.

5.3 Datenbankabstraktion

5.3.1 Temporäre Tabellen

Zum Kopieren ganzer Collections wurde bisher immer eine temporäre Tabelle erstellt, in welche die zu kopierende Resourcen kopiert wurden. Dabei wurden gleich alle veränderten Live-Properties und URI neu gesetzt und anschließend aus dieser temporären Tabelle alle Einträge in die richtige Tabelle übernommen. Schematisch sind das drei Schritte bzw. SQL Statements.

1. Prüfen ob temporäre Tabelle noch vorhanden, gegebenenfalls löschen

2. Temporäre Tabelle erstellen, mit zu kopierenden Resourcen initialisieren, URI und Live-Properties setzten.
3. Angepasste Resourcen zurückschreiben.

Jeder einzelner SQL Statement ist eine zeit- und rechenintensive Operation, die wenn möglich, vermieden werden sollte. Der Umweg über die temporären Tabellen lässt sich einfach auf einen SQL Befehl reduzieren, dieser dann direkt auf die Zieltabelle angewendet wird. Die Initialisierung der Tabelle in `CREATE TEMPORARY TABLE` Statements findet mittels eines `SELECT` Statements statt. Einzelne Spalten des Statements werden durch Konstanten oder Ergebnisse aus Ausdrücke ersetzt. Genau dieses `SELECT` Statement kann direkt im `INSERT` Statement verwendet werden. Das Zwischenspeichern in einer temporären Tabelle ist somit nicht mehr notwendig und es müssen zwei SQL Statements weniger ausgeführt werden.

5.3.2 Volltextsuche

Da die Volltextsuche über eine SQL Funktion abstrahiert wird, sind drei Aufgaben zu erledigen. Die erste ist die Definition eines Funktionsnamen samt Funktionsparameter und Rückgabetyps. Diese Funktion wird später im SQL Statement anstelle der MySQL spezifischen SQL Syntax aufgerufen.

Listing 5.1 zeigt die Funktionssignatur wie sie in Catacomb erwartet wird. Der erste Parameter enthält die Id der Resource und der zweite den Text, zudem der Relevanzwert, der berechnet werden soll. Rückgabewert ist der Relevanzwert.

```
1 FT_SEARCH_TEXTCONTENT( id INTEGER, txt VARCHAR) RETURNS DOUBLE
```

Listing 5.1: UDF ft_search_textcontent() Funktions Signatur

Als zweites ist in `search.c` die Funktion `parse_where()` anzupassen. Hier muss anstatt der MySQL Syntax die eben definierte Funktion in das SQL Statement integriert werden.

Der dritte Punkt ist das datenbankspezifische Implementieren der Funktion. Die Listings A.1 und A.2 aus dem Anhang zeigen beispielhaft die Implementierung, jeweils für MySQL und PostgreSQL.

5.3.3 Generierte Id abrufen

Die von der Datenbank generierten Id wird in der bisherigen Implementation über die MySQL API abgerufen. Zur Abstraktion dieser wird es eine neue C Funktion `dbms_insert_id()` geben. Diese Funktion hat als Funktionsparameter einen Zeiger auf die Datenbank Handle und den Tabellennamen und Spaltennamen. Der Tabellennamen und Spaltennamen werden für die SQL Abfrage benötigt.

Innerhalb der Funktion wird eine einfache SQL Abfrage auf auf die eigene SQL Funktion `LAST_INSERT_ID_BY_TABLENAME_AND_COLUMN(table, column)` gestartet.

Der komplette Quellcode ist unter Listing A.5 im Anhang zu finden.

5.4 Transaktionen

5.4.1 Catacomb Anpassung

Damit Catacomb korrekt mit Transaktionen funktionieren, darf im Regelfall kein Statement fehlschlagen. In der Analyse hat sich ein Problem ergeben, dass im Regelfall zu Transaktions abbrüchen führt. Zum Kopieren ganzer Collections, wird in der Funktion `dbms_copy_resource()` die betreffende Resourcen in eine temporäre Tabelle kopiert. Da diese Tabelle einen festen Namen verwendet, wird vor den eigentlichen immer ein `DROP` Statement ausgeführt. In der Regel wird diese Operation fehlschlagen. Wenn diese Operation innerhalb einer Transaktion stattfindet, löst es einen Datenbank Rollback aus.

5.4.2 mod_dav Erweiterung

Aus der Analyse und dem Entwurf hat sich ergeben, dass die Transaktionsverarbeitung so früh wie möglich stattfinden soll. Dafür bietet es sich an, die mod_dav Provider Interfaces, welches Catacomb implementiert, um ein weiteres Interface für Transaktionen zu erweitern.

Die Implementierung dieser Hooks folgt im wesentlichen immer nach dem gleichen Schema. Weshalb nun eine kurze Einführung in die Apache Module Programmierung gegeben wird.

Jedes Apache Module beginnt mit dem Exportieren einer Module Datenstruktur. Mit dieser Struktur gibt man sich Apache als Module zu erkennen und definiert einen Einsprungspunkt für die weiterführende Konfiguration. Nachfolgend werden die einzelnen Elemente der Struktur der Reihe nach kurz erläutert.

```
1  module AP_MODULE_DECLARE_DATA modulename =
2  {
3      STANDARD20_MODULE_STUFF,
4      dir_conf,
5      merge_dir_conf,
6      server_conf,
7      merge_server_conf,
8      cmd_table,
9      register_hooks
10 }
```

Listing 5.2: Apache Moduldeklaration

1. `STANDARD20_MODULE_STUFF` ist ein Makro, welches unteranderem sicherstellt, dass das kompilierte Module nur in binärkompatiblen Servern geladen wird, indem es Versionsinformationen bereitstellt.
2. Verzeichniskonfiguration: enhält einen Zeiger auf eine Funktion die `<Directory>`, `<Files>` und `<Location>` Umgebungen in der Konfigurationsdatei verarbeitet.
3. Verzeichniskonfiguration zusammenführen: Zeiger auf eine Funktion die verschachtelte Verzeichniskonfigurationen zusammenführt. D.h. welcher Wert aus Einstellungen hat vorrang.
4. Serverkonfiguration: Funktionszeiger für Konfigurationsdirektive die Server bzw. Virtuelle Server betreffen.
5. Serverkonfiguration zusammenführen: Funktionszeiger auf eine Funktion die Serverkonfigurationen zusammenführt.
6. Befehlstabelle: Enthält eine Liste aller Direktiven, die dieses Module implementiert.
7. Hook Registierung: enthält Funktionszeiger zur Registierung der vom Module implementierten Hooks.

Zur Registierung von Funktionen an Standard Hooks stellt Apache Funktionen zur Verfügung, mit deren Hilfe auf die Ausführungsreihenfolge Einfluss genommen werden kann. Listing 5.3 zeigt die Hook Registierungsfunktion in Apache für Handler. Diese Funktionen folgen dem Schema `ap_hook_hookname()`.

```
1 ap_hook_handler(helloworld_handler, NULL, NULL, APR_HOOK_MIDDLE);
```

Listing 5.3: Funktion zur Hook Registierung

1. Funktionszeiger auf die Funktion die diesen Hook implementiert.
2. Erlaubt das Definieren eines Vorgängers, der zuvor ausprobiert werden muss.
3. Erlaubt das Definieren eines Nachfolgers, der nachfolgend ausprobiert werden muss.
4. Definiert, wo sich die Funktion innerhalb ihres Hooks einordnen soll. In der Regel verwendet man die Konstante `APR_HOOK_MIDDLE`, Sie besagt, dass die Funktion weder am Anfang, noch am Schluss einer Anfragenbearbeitung aufgerufen wird.

Wichtige Hooks, die man häufiger in der Apache Module Entwicklung antreffen wird, sind:

handler Verarbeitet Anfrage und erzeugt eine HTTP Antwort.

fixups Last-Minute Änderungen bevor die eigentliche Verarbeitung durch den Handler stattfindet.

insert_filter Die vom Handler erzeugte Antwort kann hier nachbearbeitet werden.

Standard Hooks haben nur einen Parameter des Typs `request_req`. Diese Struktur enthält alle Informationen einer HTTP Anfrage und deren Bearbeitung. Zum Beispiel die verwendete HTTP Methode, die angeforderte Resource (URI) und den aktuellen HTTP Status der Anfrage.

Um die Konfigurationsdatei um eigene Direktiven erweitern zu können, existiert die Befehlstabelle in der Moduldeklaration. Apache stellt Makros zur Verfügung die die Deklaration der Tabelle vereinfachen. Listing 5.4 zeigt exemplarisch solch eine Tabelle.

```
1  static const command_rec dav_repos_cmds[] =
2  {
3      AP_INIT_TAKE1("DavDBMSTmpDir", dav_repos_tmp_dir_cmd, NULL, RSRC_CONF,
4                    "specify the MYSQL_TMP_DIR for a directory or location"),
5
6      AP_INIT_TAKE2("DavDBMSFileDir", dav_repos_file_dir_cmd, NULL, RSRC_CONF,
7                    "specify the directory for permanent external storage"),
8
9      {NULL}
10 };
```

Listing 5.4: Beispiel Apache Befehlstabelle

Alle Makros für die Befehlstabelle erwarten 5 Parameter.

1. Den Namen wie er in der Konfigurationsdatei erscheint.
2. Funktionszeiger auf eine Funktion die den Parameter auswertet.
3. Zeiger auf eine Datenstruktur, meist nicht verwendet.
4. Kontext wann diese Direktive erlaubt ist, z.B. `RSRC_CONF` als Kontext darf nur auf der untersten Ebene und in VirtualHosts verwendet werden.
5. Enthält eine kurze Hilfe Nachricht für diese Direktive.

5.4.3 Implementation mod_dav Transaktion Hook

Die Implementation ist zweigeteilt, der erste Teil ist die Implementierung des neu definierten Interface, das Füllen der Datenstruktur mit den Funktionszeigern und der Definition der Datenstruktur `dav_transaction_private`. Der zweite Teil besteht aus dem Aufruf der Datenbankfunktionen zum starten und beenden einer Transaktion.

5.5 Datenbank & SQL

5.5.1 Benötigte SQL Befehle

Bei einer Implementation für nur eine Datenbank kümmert man sich in der Regel nicht darum wie andere Datenbanken etwas syntaktisch umsetzen. Selbst wenn man sich an einen SQL Standard hält, ist es keine Garantie, dass es auf allen Datenbanken läuft.

Deshalb muss immer geprüft werden, ob die gewünschte Datenbank die benötigten SQL Konstrukte und Funktionen unterstützt. Das Listing 5.5 zeigt welche SQL Funktionen und Befehle Catacomb verwendet.

```
1 SUBSTRING(str, len)
2 INSERT INTO tabelle (spalten) SELECT spalten FROM tabelle WHERE where_condition
3 SELECT {spalten} FROM tabelle WHERE where_condition
4 INSERT INTO (spalten) VALUES ({ausdruck})
5 UPDATE tabelle SET {spalte = wert} WHERE where_condition
```

Listing 5.5: Benötigte SQL Funktionen und Befehle

5.5.2 UDF - User Definied Functions

Die Umsetzung dieses Abstraktionskonzepts beruht zu großen Teilen auf der Fähigkeit der Datenbank, Benutzer eigene Funktionen erstellen zu lassen. Es bietet den großen Vorteil Datenbankspezifische Funkionalität in die Datenbank auslagern zukönnen. Der SQL Standard definiert eine rudimentäre Sprache für den Funktionsinhalt. Diese muss nicht verwendet werden, sofern die Sprache von der Datenbank unterstützt wird. Stattdessen kann für jede Datenbank eine herstellerspezifische verwendet werden, die den Zugriff auf die volle Funkionalität der Datenbank ermöglicht. Häufig können auch Programmiersprachen aus der normalen Programmierung verwendet werden, die zusätzlich zum normalen Sprachumfang datenbankspezifische Erweiterungen beinhalten. Zum Beispiel kann PostgreSQL, neben der eigenen an SQL angelehnten Sprache PL/pgSQL, auch die Programmiersprachen Perl, Python und TCL verwenden.

5.5.3 Tabellen Definitionen

Die Definition der Tabellenstruktur ist in der Regel nur einmal bei der Installation des Servers notwendig. Zudem muss sich nicht das Programm darum kümmern, dass diese Tabellenstrukturen angelegt werden. Daher stellt es hier kein Problem dar, wenn datenbankspezifische Syntax verwendet wird. Zwar ist auch die Tabellendefinition im SQL Standard enthalten und die meisten Datenbanken orientieren sich an dieser Syntax. Aber so gut wie alle verwenden zusätzlich noch Hersteller spezifische Erweeiterungen die mal mehr oder weniger obligatorisch sind.

Durch das Erlauben der herstellerspezifischen Erweiterungen bzw. Variationen muss zwar für jede Datenbank ein eigenes SQL Skript erstellt werden. Dafür können z.B. die Datentypen selber bestimmt werden, Autowerte für Ids verwendet werden, anstatt über Sequenzen zu gehen oder in MySQL die verwendetete Storage Engine beeinflussen.

6 Evaluation

Die Evaluation hat das Ziel die Implemetation mit der bisherigen Version zu vergleichen und Unterschiede in Konformität zum WebDAV Standard zu erkennen und analysieren. Ebenso wird die Performance unter den Versionen verglichen und analysiert.

6.1 Standardkonformität WebDAV

Damit eine möglichst hohe Interoperabilität zu vielen WebDAV Clients gewährleistet werden kann, ist es notwendig die Implementierung gegen den WebDAV Standard zu testen. Dazu existiert die «WebDAV server protocol compliance test suite» Litmus. Sie testet eine Server Implementation gegen den WebDAV Standard RFC2518 [Weba].

Eingesetzt wird die Litmus Version 0.11 vom 23. Janunar 2007.

Da diese Diplomarbeit den Catacomb nicht um neue WebDAV Features erweitert, ist es nur das Ziel mindestens denselben Level an WebDAV Standard Übereinstimmung zu erreichen. Dabei sind jeweils der Catacomb mit und ohne ACP Erweiterung zu testen und mit den Ergebnissen der Version mit apr_dbd Datenbankanbindung zu vergleichen.

Wie die Logdateien für den Catacomb in der Version 0.9.4 zeigt Listing B.1 und die Logdatei für den Catacomb mit ACP Erweiterung zeigt Listing B.3 im Anhang. Sowohl die Version nur mit apr_dbd Datenbankanbindung also auch die Version mit ACP Protokoll und apr_dbd Datenbankanbindung haben jeweils die exakt gleiche Übereinstimmung mit dem WebDAV Standard wie die jeweilige Version ohne apr_dbd Datenbankanbindung wie Listings B.2 und B.4 zeigen.

6.2 Geschwindigkeit WebDAV

An dieser Stelle soll die Geschwindigkeit der Implementation verglichen werden und auftretende große Unterschiede genauer analysiert werden. Dazu existiert das «WebDAV Server Performance Test Suite» Prestan. Prestan erstellt auf dem Server Testdaten und führt eine Reihe von Testfällen aus und misst die Zeit die dafür benötigt wurde.

Die Testergebnisse wurden alle auf demselben Rechner unter gleichen Bedingungen erstellt. Dabei wurde Server und Client auf dem selben Rechner ausgeführt. Das hat den Vorteil, dass die Latenz und Bandbreite eines Netzwerks sich auf das Ergebnis auswirken

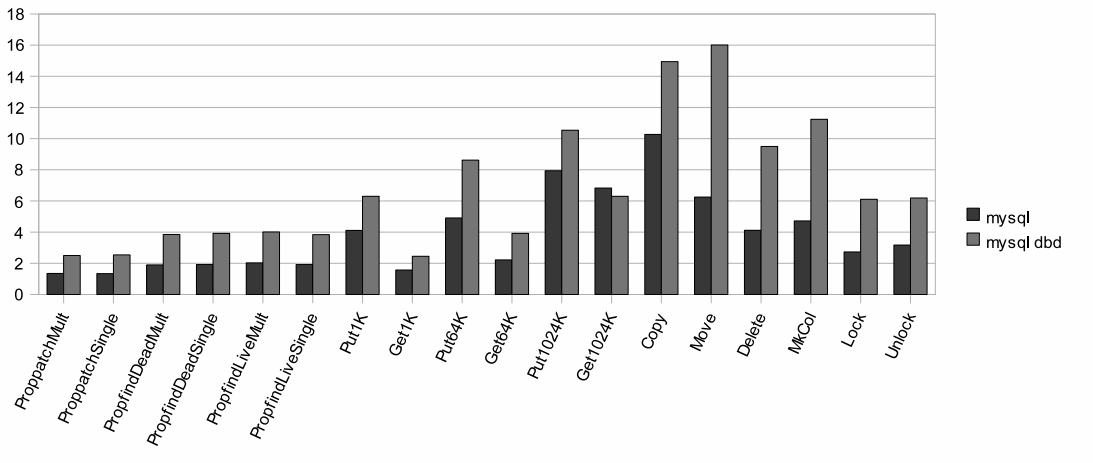

Abbildung 6.1: WebDAV Performance Diagramm 1

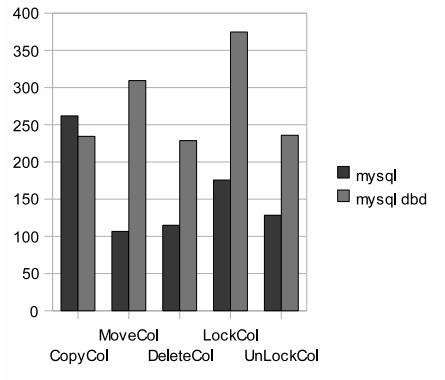

Abbildung 6.2: WebDAV Performance Diagramm 2

kann. Die zusätzliche Belastung des Rechners durch das Testprogramm Prestan ist gering, da es abgesehen vom Protokollieren der Testergebnisse keine Eingabe und Ausgabe erzeugt.

6.2.1 Implemtation ohne ACP

Als erstes wird die Implementationen ohne das Access Control Protocol miteinander verglichen. Die Tabelle 6.1 zeigt die Testergebnisse des Catacomb Server in der Version 0.9.4 und der in diese Diplomarbeit erstellten Version mit apr_dbd als Datenbankanbindung. Zusätzlich zu den Durchschnittszeiten der einzelen Testfällen, werden die Zeiten ein Verhältnis zueinander gesetzt.

Bei Betrachtung der Diagramme in Abbildungen 6.1, 6.2 und der letzen Spalte in der Ta-

	Catacomb 0.9.4 (ms)	Catacomb dbd (ms)	0.9.4/dbd
ProppatchMult	1,35	2,50	0,54
ProppatchSingle	1,34	2,54	0,53
PropfindDeadMult	1,90	3,85	0,49
PropfindDeadSingle	1,92	3,92	0,49
PropfindLiveMult	2,03	4,01	0,51
PropfindLiveSingle	1,92	3,84	0,50
Put1K	4,11	6,30	0,65
Get1K	1,57	2,45	0,64
Put64K	4,91	8,62	0,57
Get64K	2,22	3,92	0,57
Put1024K	7,94	10,54	0,75
Get1024K	6,83	6,30	1,08
Copy	10,27	14,94	0,69
Move	6,25	16,01	0,39
Delete	4,12	9,50	0,43
MkCol	4,72	11,24	0,42
CopyCol	261,96	234,47	1,12
MoveCol	106,6	309,38	0,34
DeleteCol	114,91	228,65	0,50
Lock	2,73	6,11	0,45
Unlock	3,17	6,19	0,51
LockCol	175,72	374,68	0,47
UnLockCol	128,39	235,84	0,54

Tabelle 6.1: Messwerte Catacomb

belle 6.1 fällt auf, dass in den meisten Testfällen die apr_dbd Implementation ungefähr doppelt solange Zeit benötigt wie ohne Implementation ohne apr_dbd. Eine mögliche Erklärung für dieses Verhalten liegt in der Art und Weise wie das Prepare in dbms implementiert ist. Bei einem Prepare bei apr_dbd wird an die Datenbank dass SQL Statement mit Platzhaltern an die Datenbank geschickt. Anschließend müssen nur noch die Werte für die Platzhalter an die Datenbank geschickt werden. Es kann beliebig oft weitere Werte für dieses Statement geschickt werden, ohne jedesmal ein SQL Statement zu generieren. In dbms dagegen wird jedesmal ein SQL Statement aus den Parametern zusammengebaut und an den Server geschickt. Wenn nur einmal Werte zum Vorbereiteten SQL Statement geschickt wird, ergibt sich, dass zwei Aktionen mit der Datenbank durchgeführt werden, im Gegensatz dazu wird bei dbms nur eine Aktion ausgeführt.

Bei SQL Statements, die nur einmal pro Request ausgeführt werden, könnte es sich anbieten das SQL Statement, mit den Werten zu generieren und so an die Datenbank zu übergegeben.

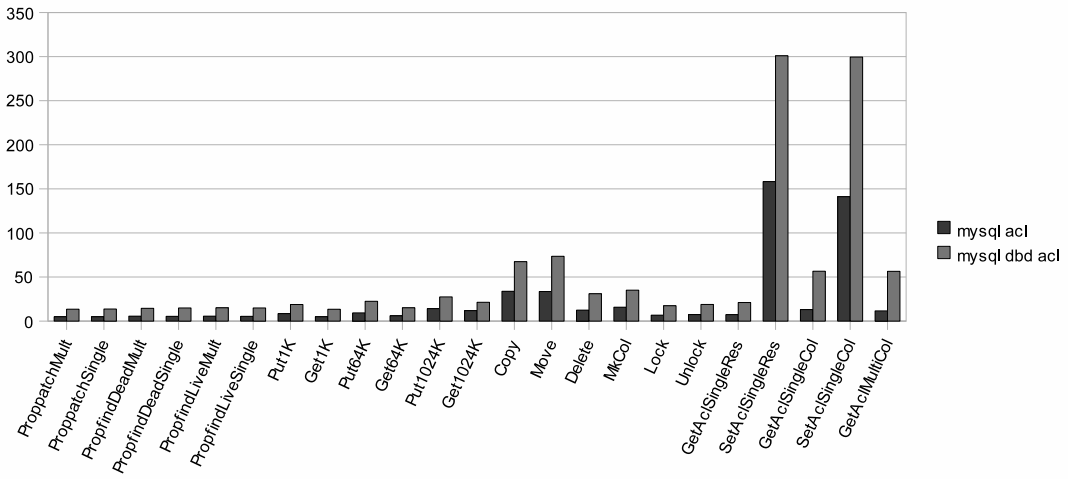

Abbildung 6.3: WebDAV Performance Diagramm ACP 1

Bei den Testfällen *PutxK*, *GetxK* und *CopyCol*, ist zu erkennen, das auf sie nicht das 1:2 Verhältnis zutrifft. Bei den beiden Testfallkategorien *PutxK* und *GetxK* ist das Aufgrund des anderen Zugriffs auf die Daten der Fall. Dabei fällt auf dass das kodieren und dekodieren der Daten in BASE64 sich nicht negativ auf die Laufzeit auswirkt.

Bei dem Testfall *CopyCol* ist der Fall anders gelagert. Hier wurde die Anzahl der SQL Statements die insgesamt ausgeführt werden muss optimiert. Zuvor wurde für das Kopieren von Collections extra eine Temporäre Tabelle erstellt. Dies ist in der dbd Version nicht mehr notwendig.

6.2.2 Implementation mit ACP

Beim Testen der Implementationen mit dem Access Control Protocol gelten selbstverständlich auch die Aussagen aus der Geschwindigkeitsanalyse ohne das ACP. Sie müssen sich aber nicht im selben Ausmaß auf die Testergebnisse auswirken, wegen des Overheads durch das ACP.

Die Tabelle 6.2 zeigt die Testergebnisse des Catacomb Server mit dem Access Control Protocol und der in diese Diplomarbeit erstellten Version mit apr_dbd als Datenbankanbindung und ACP. Zusätzlich zu den Durchschnittszeiten der einzelen Testfällen, werden die Zeiten in ein Verhältnis zueinander gesetzt.

Die Abbildungen 6.3 und 6.4 sind die grafische Darstellung der Werte aus der Tabelle 6.2.

Auffallend sind die Testfälle *CopyCol*, *MoveCol*, *DeleteCol* und *SetAclxCol*. Da in diesen Testfällen die Operation auf Collections angewendet wird, sieht die Implementation vor

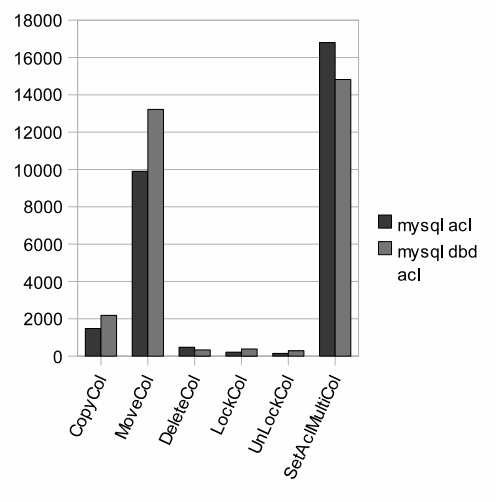

Abbildung 6.4: WebDAV Performance Diagramm ACP 2

die ACLs jeder Resource und Collection nacheinander zu modifizieren. In genau diesem Fall bringt das explizite Vorbereiten von SQL Statements einen Laufzeitvorteil.

	Catacomb ACP (ms)	Catacomb ACP dbd (ms)	0.9.4/dbd
ProppatchMult	4,97	13,54	0,37
ProppatchSingle	4,94	13,68	0,36
PropfindDeadMult	5,60	14,49	0,39
PropfindDeadSingle	5,39	14,89	0,36
PropfindLiveMult	5,60	15,09	0,37
PropfindLiveSingle	5,39	14,91	0,36
Put1K	8,45	18,84	0,45
Get1K	4,94	13,47	0,37
Put64K	9,33	22,45	0,42
Get64K	6,00	15,08	0,40
Put1024K	14,13	27,46	0,51
Get1024K	11,86	21,35	0,56
Copy	33,80	67,38	0,50
Move	33,53	73,50	0,46
Delete	12,45	31,03	0,40
MkCol	15,77	35,05	0,45
CopyCol	1481,60	2183,66	0,68
MoveCol	9905,21	13220,27	0,75
DeleteCol	475,27	331,93	1,43
Lock	6,75	17,45	0,39
Unlock	7,37	18,86	0,39
LockCol	211,79	380,80	0,56
UnLockCol	145,11	285,19	0,51
GetAclSingleRes	7,33	21,05	0,35
SetAclSingleRes	158,23	300,94	0,53
GetAclSingleCol	13,07	56,61	0,23
SetAclSingleCol	141,07	299,48	0,47
GetAclMultiCol	11,6	56,47	0,21
SetAclMultiCol	16796,62	14820,78	1,13

Tabelle 6.2: Messwerte Catacomb mit ACP

7 Zusammenfassung & Ausblick

7.1 Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde die Datenbankschicht des Catacomb WebDAV Server modifiziert, um unabhängiger von einer spezifischen Datenbank zu sein. Dazu wurden erst die notwendigsten Grundlagen zum Apache Webserver, WebDAV Protokoll, Relationalen Datenbanken und Catacomb WebDAV Server eingeführt. Nachfolgend wurde das Datenbankmodell für Catacomb erläutert.

Nach den Einführungen wurde gezeigt welche Bereiche von Catacomb für eine Datenbankunabhängigkeit überarbeitet werden müssen und wie sie gestaltet werden sollen. Der nächste Teil zeigt, wie die notwendigen Änderungen umgesetzt werden.

Abschließend wird die Implementierung hinsichtlich ihrer Übereinstimmung mit dem WebDAV Standard geprüft und mit den aktuellen äquivalenten Version ohne dbd verglichen. Dasselbe wird mit dem Aspekt der Geschwindigkeit gemacht.

7.2 Ausblick

Mit der Beseitigung der Abhängigkeiten zu MySQL ist ein weiterer Meilenstein in der Catacomb Entwicklung erreicht. Nichts desto trotz gibt es noch einige Entwicklungs- und Verbesserungsmöglichkeiten.

Ein großer und wichtiger Punkt meiner Meinung nach ist es Unittests einzuführen. Ein wichtiger Aspekt dabei ist, neben den Unittests selber, auch das automatische Aufsetzen und Befüllen einer Testdatenbank mit aussagekräftigen Testdaten evtl. für verschiedene Datenbanktypen.

Das Projekt Catacomb definiert selber keinen Coding Style. Da es sich um ein Apache Modul handelt würde es sich anbieten sich an deren Style Guide zu orientieren. (http://httpd.apache.org/dev/styleguide.html)

Zur Verbesserung der Geschwindigkeit könnte man neben der Reduktion der auszuführenden SQL Statements auch auf Datenbankfeatures zurückgreifen. Wobei zu klären ist, wie weit verbreitet diese in der Relationalen Datenbankwelt sind. Beispielsweise kann ich mir vorstellen, dass mit *Delete Casacdes* im Zusammenhang mit Fremdschlüsseln oder

Stored Procedures für komplexe Datenmodifikationen messbare geschwindigkeitsvorteile zu erzielen sind.

A Quellcode

A.1 User Defined Functions

```
1  CREATE FUNCTION FT_SEARCH_TEXTCONTENT (id integer, txt varchar(255)) RETURNS
       double
2  BEGIN
3      DECLARE val double;
4
5      SELECT MATCH textcontent AGAINST ( txt ) INTO val FROM dasl_resource WHERE
           serialno = id;
6      RETURN val;
7  END;
```

Listing A.1: UDF ft_search_textcontent() für MySQL

```
1  CREATE FUNCTION public.ft_search_textcontent (id int8, txt varchar) RETURNS float4
       AS
2  $$
3      DECLARE val float4;
4  BEGIN
5
6      SELECT rank(textcontent_idx, q) INTO val
7          FROM dasl_resource, to_tsquery('default', txt) AS q WHERE serialno = id;
8
9      RETURN val;
10 END
11 $$
12 LANGUAGE 'plpgsql';
```

Listing A.2: UDF ft_search_textcontent() für PostgreSQL

```
1  CREATE FUNCTION LAST_INSERT_ID_BY_TABLENAME_AND_COLUMN(intable varchar(255),
2                                                         incolumn varchar(255))
3  RETURNS int(11)
4  RETURN LAST_INSERT_ID();
```

Listing A.3: UDF last_insert_id_by_tablename_and_column() für MySQL

```
1  CREATE FUNCTION LAST_INSERT_ID_BY_TABLENAME_AND_COLUMN(varchar,varchar ) RETURNS
       int8
2  AS $$
3  DECLARE
4  intable alias for $1;
5  incolumn alias for $2;
6  myvalue integer;
7
8  BEGIN
9
10 for myvalue in execute  ' select  currval('' '
```

```
|| intable || '_' || incolumn
|| '_seq)
' Loop
return myvalue;


END loop;
END;

$$
language 'plpgsql';
```

Listing A.4: UDF last_insert_id_by_tablename_and_column() für PostgreSQL

A.2 C Quellcode

```
long dbms_insert_id(dav_repos_db *db, const char *table, const char *column)
{
    apr_pool_t *pool = db->r->pool;
    ap_dbd_t *dbd = NULL;
    apr_dbd_results_t *res = NULL;
    apr_dbd_row_t *row = NULL;
    int rv = 0;
    long  retv = -1;
    const char *stmt = apr_psprintf(pool, "SELECT
        LAST_INSERT_ID_BY_TABLENAME_AND_COLUMN('%s', '%s')", table, column);

    dbd = dbms_get_db(db);

    if(dbd)
    {
        rv = apr_dbd_select(dbd->driver, pool, dbd->handle, &res, stmt, 1);

        /* Loop over der ResultSet */
        for(rv = apr_dbd_get_row(dbd->driver, pool, res, &row, -1);
            rv != -1;
            rv = apr_dbd_get_row(dbd->driver, pool, res, &row, -1)) {

            if(rv != 0) {
                db_error_message_new(dbd, rv, "Error while retrieving ID",
                    __func__);
            }
            else {
                retv = atol(apr_dbd_get_entry(dbd->driver, row, 0));
            }
        }
    }

    return retv;
}
```

Listing A.5: Generierte Id abrufen - dbms_insert_id()

B Litmus Testergebnisse

B.1 Catacomb ohne Access Control Protocol

```
-> running 'basic':
 0. init.................. pass
 1. begin................. pass
 2. options............... pass
 3. put_get............... pass
 4. put_get_utf8_segment.. pass
 5. mkcol_over_plain...... pass
 6. delete................ pass
 7. delete_null........... pass
 8. delete_fragment....... pass
 9. mkcol................. pass
10. mkcol_again........... pass
11. delete_coll........... pass
12. mkcol_no_parent....... pass
13. mkcol_with_body....... pass
14. finish................ pass
<- summary for 'basic': of 15 tests run: 15 passed, 0 failed. 100.0%
-> running 'copymove':
 0. init.................. pass
 1. begin................. pass
 2. copy_init............. pass
 3. copy_simple........... pass
 4. copy_overwrite........ pass
 5. copy_nodestcoll....... pass
 6. copy_cleanup.......... pass
 7. copy_coll............. pass
 8. copy_shallow.......... pass
 9. move.................. pass
10. move_coll............. pass
11. move_cleanup.......... pass
12. finish................ pass
<- summary for 'copymove': of 13 tests run: 13 passed, 0 failed. 100.0%
-> running 'props':
 0. init.................. pass
 1. begin................. pass
 2. propfind_invalid...... pass
 3. propfind_invalid2..... pass
 4. propfind_d0........... pass
 5. propinit.............. pass
 6. propset............... pass
 7. propget............... pass
 8. propextended.......... pass
 9. propmove.............. pass
10. propget............... pass
11. propdeletes........... pass
12. propget............... pass
13. propreplace........... pass
14. propget............... pass
```

```
 15. propnullns..... pass
 16. propget..... pass
 17. prophighunicode..... pass
 18. propget..... pass
 19. propremoveset..... pass
 20. propget..... pass
 21. propsetremove..... pass
 22. propget..... pass
 23. propvalnspace..... pass
 24. propwformed..... pass
 25. propinit..... pass
 26. propmanyns..... pass
 27. propget..... pass
 28. propcleanup..... pass
 29. finish..... pass
<- summary for 'props': of 30 tests run: 30 passed, 0 failed. 100.0%
-> running 'locks':
  0. init..... pass
  1. begin..... pass
  2. options..... pass
  3. precond..... pass
  4. init_locks..... pass
  5. put..... pass
  6. lock_excl..... pass
  7. discover..... pass
  8. refresh..... pass
  9. notowner_modify..... pass
 10. notowner_lock..... pass
 11. owner_modify..... pass
 12. notowner_modify..... pass
 13. notowner_lock..... pass
 14. copy..... pass
 15. cond_put..... pass
 16. fail_cond_put..... pass
 17. cond_put_with_not..... pass
 18. cond_put_corrupt_token WARNING: PUT failed with 400 not 423
     ..... pass (with 1 warning)
 19. complex_cond_put..... pass
 20. fail_complex_cond_put. pass
 21. unlock..... pass
 22. fail_cond_put_unlocked pass
 23. lock_shared..... pass
 24. notowner_modify..... pass
 25. notowner_lock..... pass
 26. owner_modify..... pass
 27. double_sharedlock..... pass
 28. notowner_modify..... pass
 29. notowner_lock..... pass
 30. unlock..... pass
 31. prep_collection..... pass
 32. lock_collection..... pass
 33. owner_modify..... pass
 34. notowner_modify..... pass
 35. refresh..... pass
 36. indirect_refresh..... FAIL (indirect refresh LOCK on /repos/litmus/lockcoll/
     via /repos/litmus/lockcoll/lockme.txt: Could not read status line: connection
     was closed by server.)
 37. unlock..... pass
 38. finish..... pass
<- summary for 'locks': of 39 tests run: 38 passed, 1 failed. 97.4%
-> 1 warning was issued.
-> running 'http':
```

```
 0. init.................. pass
 1. begin................. pass
 2. expect100............. pass
 3. finish................ pass
<- summary for 'http': of 4 tests run: 4 passed, 0 failed. 100.0%
```

Listing B.1: Litmus Logdatei für Catacomb ohne Access Control Protocol

B.2 Catacomb ohne Access Control Protocol und apr_dbd

```
-> running 'basic':
 0. init.................. pass
 1. begin................. pass
 2. options............... pass
 3. put_get............... pass
 4. put_get_utf8_segment.. pass
 5. mkcol_over_plain...... pass
 6. delete................ pass
 7. delete_null........... pass
 8. delete_fragment....... pass
 9. mkcol................. pass
10. mkcol_again........... pass
11. delete_coll........... pass
12. mkcol_no_parent....... pass
13. mkcol_with_body....... pass
14. finish................ pass
<- summary for 'basic': of 15 tests run: 15 passed, 0 failed. 100.0%
-> running 'copymove':
 0. init.................. pass
 1. begin................. pass
 2. copy_init............. pass
 3. copy_simple........... pass
 4. copy_overwrite........ pass
 5. copy_nodestcoll....... pass
 6. copy_cleanup.......... pass
 7. copy_coll............. pass
 8. copy_shallow.......... pass
 9. move.................. pass
10. move_coll............. pass
11. move_cleanup.......... pass
12. finish................ pass
<- summary for 'copymove': of 13 tests run: 13 passed, 0 failed. 100.0%
-> running 'props':
 0. init.................. pass
 1. begin................. pass
 2. propfind_invalid...... pass
 3. propfind_invalid2..... pass
 4. propfind_d0........... pass
 5. propinit.............. pass
 6. propset............... pass
 7. propget............... pass
 8. propextended.......... pass
 9. propmove.............. pass
10. propget............... pass
11. propdeletes........... pass
12. propget............... pass
13. propreplace........... pass
14. propget............... pass
15. propnullns............ pass
16. propget............... pass
```

```
17. prophighunicode....... pass
18. propget............... pass
19. propremoveset......... pass
20. propget............... pass
21. propsetremove......... pass
22. propget............... pass
23. propvalnspace......... pass
24. propwformed........... pass
25. propinit.............. pass
26. propmanyns............ pass
27. propget............... pass
28. propcleanup........... pass
29. finish................ pass
<- summary for ‘props’: of 30 tests run: 30 passed, 0 failed. 100.0%
-> running ‘locks’:
 0. init.................. pass
 1. begin................. pass
 2. options............... pass
 3. precond............... pass
 4. init_locks............ pass
 5. put................... pass
 6. lock_excl............. pass
 7. discover.............. pass
 8. refresh............... pass
 9. notowner_modify....... pass
10. notowner_lock......... pass
11. owner_modify.......... pass
12. notowner_modify....... pass
13. notowner_lock......... pass
14. copy.................. pass
15. cond_put.............. pass
16. fail_cond_put......... pass
17. cond_put_with_not..... pass
18. cond_put_corrupt_token WARNING: PUT failed with 400 not 423
    ..................... pass (with 1 warning)
19. complex_cond_put...... pass
20. fail_complex_cond_put. pass
21. unlock................ pass
22. fail_cond_put_unlocked pass
23. lock_shared........... pass
24. notowner_modify....... pass
25. notowner_lock......... pass
26. owner_modify.......... pass
27. double_sharedlock..... pass
28. notowner_modify....... pass
29. notowner_lock......... pass
30. unlock................ pass
31. prep_collection....... pass
32. lock_collection....... pass
33. owner_modify.......... pass
34. notowner_modify....... pass
35. refresh............... pass
36. indirect_refresh...... FAIL (indirect refresh LOCK on /repos/litmus/lockcoll/
    via /repos/litmus/lockcoll/lockme.txt: Could not read status line: connection
    was closed by server.)
37. unlock................ pass
38. finish................ pass
<- summary for ‘locks’: of 39 tests run: 38 passed, 1 failed. 97.4%
-> 1 warning was issued.
-> running ‘http’:
 0. init.................. pass
 1. begin................. pass
```

```
 2. expect100.............. pass
 3. finish................. pass
<- summary for 'http': of 4 tests run: 4 passed, 0 failed. 100.0%
```

Listing B.2: Litmus Logdatei für Catacomb ohne Access Control Protocol und apr_dbd

B.3 Catacomb mit Access Control Protocol

```
-> running 'basic':
 0. init.................. pass
 1. begin................. pass
 2. options............... pass
 3. put_get............... pass
 4. put_get_utf8_segment.. pass
 5. mkcol_over_plain...... pass
 6. delete................ pass
 7. delete_null........... pass
 8. delete_fragment....... pass
 9. mkcol................. pass
10. mkcol_again........... pass
11. delete_coll........... pass
12. mkcol_no_parent....... pass
13. mkcol_with_body....... pass
14. finish................ pass
<- summary for 'basic': of 15 tests run: 15 passed, 0 failed. 100.0%
-> running 'copymove':
 0. init.................. pass
 1. begin................. pass
 2. copy_init............. pass
 3. copy_simple........... pass
 4. copy_overwrite........ pass
 5. copy_nodestcoll....... pass
 6. copy_cleanup.......... pass
 7. copy_coll............. pass
 8. copy_shallow.......... pass
 9. move.................. pass
10. move_coll............. pass
11. move_cleanup.......... pass
12. finish................ pass
<- summary for 'copymove': of 13 tests run: 13 passed, 0 failed. 100.0%
-> running 'props':
 0. init.................. pass
 1. begin................. pass
 2. propfind_invalid...... pass
 3. propfind_invalid2..... pass
 4. propfind_d0........... pass
 5. propinit.............. pass
 6. propset............... pass
 7. propget............... pass
 8. propextended.......... pass
 9. propmove.............. pass
10. propget............... pass
11. propdeletes........... pass
12. propget............... pass
13. propreplace........... pass
14. propget............... pass
15. propnullns............ pass
16. propget............... pass
17. prophighunicode....... pass
18. propget............... pass
```

```
 19. propremoveset......... pass
 20. propget............... pass
 21. propsetremove......... pass
 22. propget............... pass
 23. propvalnspace......... pass
 24. propwformed........... pass
 25. propinit.............. pass
 26. propmanyns............ pass
 27. propget............... pass
 28. propcleanup........... pass
 29. finish................ pass
<- summary for ‘props’: of 30 tests run: 30 passed, 0 failed. 100.0%
-> running ‘locks’:
  0. init.................. pass
  1. begin................. pass
  2. options............... pass
  3. precond............... pass
  4. init_locks............ pass
  5. put................... pass
  6. lock_excl............. pass
  7. discover.............. pass
  8. refresh............... pass
  9. notowner_modify....... pass
 10. notowner_lock......... pass
 11. owner_modify.......... pass
 12. notowner_modify....... pass
 13. notowner_lock......... pass
 14. copy.................. pass
 15. cond_put.............. pass
 16. fail_cond_put......... WARNING: PUT failed with 423 not 412
     ..................... pass (with 1 warning)
 17. cond_put_with_not..... pass
 18. cond_put_corrupt_token WARNING: PUT failed with 400 not 423
     ..................... pass (with 1 warning)
 19. complex_cond_put...... pass
 20. fail_complex_cond_put. pass
 21. unlock................ pass
 22. fail_cond_put_unlocked FAIL (conditional PUT with invalid lock-token should
     fail: 204 No Content)
 23. lock_shared........... pass
 24. notowner_modify....... pass
 25. notowner_lock......... pass
 26. owner_modify.......... pass
 27. double_sharedlock..... pass
 28. notowner_modify....... pass
 29. notowner_lock......... pass
 30. unlock................ pass
 31. prep_collection....... pass
 32. lock_collection....... pass
 33. owner_modify.......... pass
 34. notowner_modify....... pass
 35. refresh............... pass
 36. indirect_refresh...... FAIL (indirect refresh LOCK on /repos/litmus/lockcoll/
     via /repos/litmus/lockcoll/lockme.txt: Could not read status line: connection
     was closed by server.)
 37. unlock................ pass
 38. finish................ pass
<- summary for ‘locks’: of 39 tests run: 37 passed, 2 failed. 94.9%
-> 2 warnings were issued.
-> running ‘http’:
  0. init.................. pass
  1. begin................. pass
```

```
 2. expect100.............. pass
 3. finish................. pass
<- summary for 'http': of 4 tests run: 4 passed, 0 failed. 100.0%
```

Listing B.3: Litmus Logdatei für Catacomb mit Access Control Protocol

B.4 Catacomb mit Access Control Protocol und apr_dbd

```
-> running 'basic':
 0. init.................. pass
 1. begin................. pass
 2. options............... pass
 3. put_get............... pass
 4. put_get_utf8_segment.. pass
 5. mkcol_over_plain...... pass
 6. delete................ pass
 7. delete_null........... pass
 8. delete_fragment....... pass
 9. mkcol................. pass
10. mkcol_again........... pass
11. delete_coll........... pass
12. mkcol_no_parent....... pass
13. mkcol_with_body....... pass
14. finish................ pass
<- summary for 'basic': of 15 tests run: 15 passed, 0 failed. 100.0%
-> running 'copymove':
 0. init.................. pass
 1. begin................. pass
 2. copy_init............. pass
 3. copy_simple........... pass
 4. copy_overwrite........ pass
 5. copy_nodestcoll....... pass
 6. copy_cleanup.......... pass
 7. copy_coll............. pass
 8. copy_shallow.......... pass
 9. move.................. pass
10. move_coll............. pass
11. move_cleanup.......... pass
12. finish................ pass
<- summary for 'copymove': of 13 tests run: 13 passed, 0 failed. 100.0%
-> running 'props':
 0. init.................. pass
 1. begin................. pass
 2. propfind_invalid...... pass
 3. propfind_invalid2..... pass
 4. propfind_d0........... pass
 5. propinit.............. pass
 6. propset............... pass
 7. propget............... pass
 8. propextended.......... pass
 9. propmove.............. pass
10. propget............... pass
11. propdeletes........... pass
12. propget............... pass
13. propreplace........... pass
14. propget............... pass
15. propnullns............ pass
16. propget............... pass
17. prophighunicode....... pass
18. propget............... pass
```

```
 19. propremoveset......... pass
 20. propget............... pass
 21. propsetremove......... pass
 22. propget............... pass
 23. propvalnspace......... pass
 24. propwformed........... pass
 25. propinit.............. pass
 26. propmanyns............ pass
 27. propget............... pass
 28. propcleanup........... pass
 29. finish................ pass
<- summary for 'props': of 30 tests run: 30 passed, 0 failed. 100.0%
-> running 'locks':
  0. init.................. pass
  1. begin................. pass
  2. options............... pass
  3. precond............... pass
  4. init_locks............ pass
  5. put................... pass
  6. lock_excl............. pass
  7. discover.............. pass
  8. refresh............... pass
  9. notowner_modify....... pass
 10. notowner_lock......... pass
 11. owner_modify.......... pass
 12. notowner_modify....... pass
 13. notowner_lock......... pass
 14. copy.................. pass
 15. cond_put.............. pass
 16. fail_cond_put......... WARNING: PUT failed with 423 not 412
     ..................... pass (with 1 warning)
 17. cond_put_with_not..... pass
 18. cond_put_corrupt_token WARNING: PUT failed with 400 not 423
     ..................... pass (with 1 warning)
 19. complex_cond_put...... pass
 20. fail_complex_cond_put. pass
 21. unlock................ pass
 22. fail_cond_put_unlocked FAIL (conditional PUT with invalid lock-token should
     fail: 204 No Content)
 23. lock_shared........... pass
 24. notowner_modify....... pass
 25. notowner_lock......... pass
 26. owner_modify.......... pass
 27. double_sharedlock..... pass
 28. notowner_modify....... pass
 29. notowner_lock......... pass
 30. unlock................ pass
 31. prep_collection....... pass
 32. lock_collection....... pass
 33. owner_modify.......... pass
 34. notowner_modify....... pass
 35. refresh............... pass
 36. indirect_refresh...... FAIL (indirect refresh LOCK on /repos/litmus/lockcoll/
     via /repos/litmus/lockcoll/lockme.txt: Could not read status line: connection
     was closed by server.)
 37. unlock................ pass
 38. finish................ pass
<- summary for 'locks': of 39 tests run: 37 passed, 2 failed. 94.9%
-> 2 warnings were issued.
-> running 'http':
  0. init.................. pass
  1. begin................. pass
```

```
112  2. expect100.............. pass
113  3. finish................. pass
114 <- summary for 'http': of 4 tests run: 4 passed, 0 failed. 100.0%
```

Listing B.4: Litmus Logdatei für Catacomb mit Access Control Protocol und apr_dbd

C Prestan Testergebnisse

C.1 Catacomb 0.9.4

```
1
2
3  Prestan, Version 0.3.0
4  Copyright(c) 2003 Teng Xu, GRASE Research Group at UCSC
5  Copyright(c) 2007 Markus Litz, Modified by DLR - 2007
6  http://www.dlr.de/sc
7
8  Server Warming Up.....................Please Wait.......Done
9
10 Start Testing http://localhost:8080/repos:
11
12
13           *********************************
14
15           * Number of Requests      10
16
17           * Number of Dead Properties   10
18
19           * Depth of Collection     10
20
21           * Width of Collection     100
22
23           * Type of Methods     WebDAV
24
25           *********************************
26
27
28
29 ProppatchMult................. Rsp = 1351 [us]
30
31 ProppatchSingle.............. Rsp = 1335 [us]
32
33 PropfindDeadMult............. Rsp = 1896 [us]
34
35 PropfindDeadSingle........... Rsp = 1915 [us]
36
37 PropfindLiveMult............. Rsp = 2032 [us]
38
39 PropfindLiveSingle........... Rsp = 1919 [us]
40
41 Put1K........................ Rsp = 4114 [us]
42
43 Get1K........................ Rsp = 1572 [us]
44
45 Put64K....................... Rsp = 4910 [us]
46
47 Get64K....................... Rsp = 2223 [us]
48
```

```
Put1024K..................... Rsp = 7941 [us]

Get1024K..................... Rsp = 6833 [us]

Copy......................... Rsp = 10272 [us]

Move......................... Rsp = 6248 [us]

Delete....................... Rsp = 4124 [us]

MkCol........................ Rsp = 4724 [us]

CopyCol...................... Rsp = 261964 [us]

MoveCol...................... Rsp = 106602 [us]

DeleteCol.................... Rsp = 114912 [us]

Lock......................... Rsp = 2732 [us]

UnLock....................... Rsp = 3166 [us]

LockCol...................... Rsp = 175721 [us]

UnLockCol.................... Rsp = 128392 [us]

ACL Not Implemented
```

Listing C.1: Prestan Logdatei in den Standardeinstellungen für Catacomb 0.9.4

C.2 Catacomb 0.9.4 mit dbd

```


Prestan, Version 0.3.0
Copyright(c) 2003 Teng Xu, GRASE Research Group at UCSC
Copyright(c) 2007 Markus Litz, Modified by DLR - 2007
http://www.dlr.de/sc

Server Warming Up.....................Please Wait.......Done

Start Testing http://localhost:8080/repos:


          *********************************

          * Number of Requests      10

          * Number of Dead Properties   10

          * Depth of Collection     10

          * Width of Collection     100

          * Type of Methods     WebDAV

          *********************************


```

```
29 ProppatchMult.................. Rsp = 2495 [us]
30
31 ProppatchSingle.................. Rsp = 2542 [us]
32
33 PropfindDeadMult.............. Rsp = 3852 [us]
34
35 PropfindDeadSingle............ Rsp = 3919 [us]
36
37 PropfindLiveMult.............. Rsp = 4012 [us]
38
39 PropfindLiveSingle............ Rsp = 3843 [us]
40
41 Put1K......................... Rsp = 6296 [us]
42
43 Get1K......................... Rsp = 2448 [us]
44
45 Put64K........................ Rsp = 8621 [us]
46
47 Get64K........................ Rsp = 3919 [us]
48
49 Put1024K...................... Rsp = 10544 [us]
50
51 Get1024K...................... Rsp = 6298 [us]
52
53 Copy.......................... Rsp = 14935 [us]
54
55 Move.......................... Rsp = 16013 [us]
56
57 Delete........................ Rsp = 9496 [us]
58
59 MkCol......................... Rsp = 11238 [us]
60
61 CopyCol....................... Rsp = 234465 [us]
62
63 MoveCol....................... Rsp = 309384 [us]
64
65 DeleteCol..................... Rsp = 228652 [us]
66
67 Lock.......................... Rsp = 6113 [us]
68
69 UnLock........................ Rsp = 6192 [us]
70
71 LockCol....................... Rsp = 374678 [us]
72
73 UnLockCol..................... Rsp = 235840 [us]
74
75 ACL Not Implemented
```

Listing C.2: Prestan Logdatei in den Standardeinstellungen für Catacomb 0.9.4 mit dbd

C.3 Catacomb ACP Branch

```
1
2
3 Prestan, Version 0.3.0
4 Copyright(c) 2003 Teng Xu, GRASE Research Group at UCSC
5 Copyright(c) 2007 Markus Litz, Modified by DLR - 2007
6 http://www.dlr.de/sc
7
8 Server Warming Up.....................Please Wait.......Done
```

```
Start Testing http://localhost:8090/repos:

          **********************************
          * Number of Requests      10
          * Number of Dead Properties   10
          * Depth of Collection     10
          * Width of Collection     100
          * Type of Methods     WebDAV
          **********************************


ProppatchMult.................. Rsp = 4968 [us]
ProppatchSingle................ Rsp = 4936 [us]
PropfindDeadMult............... Rsp = 5598 [us]
PropfindDeadSingle............. Rsp = 5394 [us]
PropfindLiveMult............... Rsp = 5597 [us]
PropfindLiveSingle............. Rsp = 5392 [us]
Put1K.......................... Rsp = 8448 [us]
Get1K.......................... Rsp = 4938 [us]
Put64K......................... Rsp = 9329 [us]
Get64K......................... Rsp = 5998 [us]
Put1024K....................... Rsp = 14162 [us]
Get1024K....................... Rsp = 11859 [us]
Copy........................... Rsp = 33801 [us]
Move........................... Rsp = 33527 [us]
Delete......................... Rsp = 12453 [us]
MkCol.......................... Rsp = 15771 [us]
CopyCol........................ Rsp = 1481599 [us]
MoveCol........................ Rsp = 9905206 [us]
DeleteCol...................... Rsp = 475272 [us]
Lock........................... Rsp = 6752 [us]
UnLock......................... Rsp = 7366 [us]

```

```
71 LockCol..................... Rsp = 211789 [us]
72
73 UnLockCol..................... Rsp = 145105 [us]
74
75 GetAclSingleRes.............. Rsp = 7330 [us]
76
77 SetAclSingleRes.............. Rsp = 158229 [us]
78
79 GetAclSingleColl.............. Rsp = 13074 [us]
80
81 SetAclSingleColl.............. Rsp = 141069 [us]
82
83 GetAclMultiColl............... Rsp = 11603 [us]
84
85 SetAclMultiColl............... Rsp = 16796624 [us]
```

Listing C.3: Prestan Logdatei in den Standardeinstellungen für Catacomb ACP Branch

C.4 Catacomb ACP Branch mit dbd

```
1
2
3  Prestan, Version 0.3.0
4  Copyright(c) 2003 Teng Xu, GRASE Research Group at UCSC
5  Copyright(c) 2007 Markus Litz, Modified by DLR - 2007
6  http://www.dlr.de/sc
7
8  Server Warming Up.....................Please Wait.......Done
9
10 Start Testing http://localhost:8090/repos:
11
12
13           **********************************
14
15           * Number of Requests      10
16
17           * Number of Dead Properties   10
18
19           * Depth of Collection     10
20
21           * Width of Collection     100
22
23           * Type of Methods     WebDAV
24
25           **********************************
26
27
28
29 ProppatchMult................. Rsp = 13544 [us]
30
31 ProppatchSingle.............. Rsp = 13679 [us]
32
33 PropfindDeadMult............. Rsp = 14490 [us]
34
35 PropfindDeadSingle........... Rsp = 14892 [us]
36
37 PropfindLiveMult............. Rsp = 15088 [us]
38
39 PropfindLiveSingle........... Rsp = 14908 [us]
40
```

```
41 Put1K..................... Rsp = 18844 [us]
42
43 Get1K..................... Rsp = 13474 [us]
44
45 Put64K.................... Rsp = 22446 [us]
46
47 Get64K.................... Rsp = 15081 [us]
48
49 Put1024K.................. Rsp = 27460 [us]
50
51 Get1024K.................. Rsp = 21350 [us]
52
53 Copy...................... Rsp = 67382 [us]
54
55 Move...................... Rsp = 73499 [us]
56
57 Delete.................... Rsp = 31029 [us]
58
59 MkCol..................... Rsp = 35054 [us]
60
61 CopyCol................... Rsp = 2183660 [us]
62
63 MoveCol................... Rsp = 13220269 [us]
64
65 DeleteCol................. Rsp = 331928 [us]
66
67 Lock...................... Rsp = 17448 [us]
68
69 UnLock.................... Rsp = 18861 [us]
70
71 LockCol................... Rsp = 380802 [us]
72
73 UnLockCol................. Rsp = 285185 [us]
74
75 GetAclSingleRes............ Rsp = 21047 [us]
76
77 SetAclSingleRes............ Rsp = 300942 [us]
78
79 GetAclSingleColl........... Rsp = 56612 [us]
80
81 SetAclSingleColl........... Rsp = 299476 [us]
82
83 GetAclMultiColl............ Rsp = 56468 [us]
84
85 SetAclMultiColl............ Rsp = 14820782 [us]
```

Listing C.4: Prestan Logdatei in den Standardeinstellungen für Catacomb ACP Branch mit dbd

D Inhalt der CD

Dieser Arbeit ist eine CD beigefügt, deren Inhalt an dieser Stelle kurz erläutert werden soll. Auf der CD sind folgende Verzeichnisse zu finden:

src Enthält die letzte Version des Catacomb WebDAV Servers in den Vier Ausführungen Catacomb, Catacomb mit ACP Erweiterung, Catacomb mt mod_dbd und Catacomb mit ACP Erweiterung und mod_dbd.

litmus Enthält die Logdatei von Litmus für die vier Catacomb WebDAV Server auf der CD.

performance Enthält die Logdatei von Prestan für die vier Catacomb WebDAV Server auf der CD.

Zudem enthält es im Hauptverzeichnis die Arbeit als PDF-Dokument.

Listings

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ACP] *Rfc3744 - web distributed authoring and versioning (webdav) access control protocol.* http://webdav.org/specs/rfc3744.html.

[bas] *Rfc3548 - the base16, base32, and base64 data encodings.* http://tools.ietf.org/html/rfc3548.

[DAS] *Web distributed authoring and versioning (webdav) search.* http://greenbytes.de/tech/webdav/draft-reschke-webdav-search-latest.html.

[Del] *Rfc3253 - versioning extensions to webdav.* http://webdav.org/specs/rfc3253.html.

[Dus04] Dusseault, Lisa: *WebDAV Next-Generation Collaborative Web Authoring.* Prentice Hall, 1. edition, 2004.

[Kew07] Kew, Nick: *The Apache Modules Book - Application Development with Apache.* Prentice Hall, 1. edition, 2007.

[Lit06] Litz, Markus: *Implementierung eines Access Control Protocols in den Catacom WebDAV Server.* Diplomarbeit, Fachhochschule Südwestfalen, 2006.

[Vos00] Vossen, Gottfried: *Datenmodelle, Datenbanksprachen und Datenbankmanagmentsysteme.* Oldenbourg Verlag, 4. Auflage, 2000.

[Weba] *Rfc2518 - http extensions for web distributed authoring and versioning (webdav).* http://webdav.org/specs/rfc2518.html.

[Webb] *Rfc4918 - http extensions for web distributed authoring and versioning (webdav).* http://webdav.org/specs/rfc4918.html.