University of Applied Sciences

GEORG-SIMON-OHM
FACHHOCHSCHULE
NÜRNBERG

Diplomarbeit
von Tobias Jähnel

September 2007

Analyse von verteilten Dateisystemen und Design eines Offline-Dateisystems.

Aufgabensteller:
Prof. Dr. Peter Trommler

Zweitprüfer:
Prof. Dr. Uwe Wienkop

Zusammenfassung der Abschlussarbeit

Bearbeiter:	Tobias Jähnel, Matrikel.-Nr. 793783		
Studiengang:	Informatik, Diplom	**Schwerpunkt:**	
Erstprüfer:	Prof. Dr. Peter Trommler	**Ausgabetag:**	22.03.2007
Zweitprüfer:	Prof. Dr. Uwe Wienkop	**Abgabetag:**	30.09.2007

Durchgeführt bei Firma:	-
Betreuer/in innerhalb der Firma:	-
Abteilung und vollständige Tel.-Nr:	-

Thema der Arbeit

Analyse von verteilten Dateisystemen und Design eines Offline-Dateisystems.

Zusammenfassung

Das Angebot an verteilten Dateisystemen, die unter einer Open-Source Lizenz verfügbar sind, ist groß. Allerdings ist die Offlinefähigkeit meist nur dürftig oder gar nicht vorhanden. Die Diplomarbeit bewertet bereits bestehende offlinefähige Dateisysteme und vergleicht die eingesetzten Konzepte. Die daraus gewonnenen Erkenntnisse dienen als Basis für das Konzept eines eigenen offlinefähigen Dateisystems, das zu einem späteren Zeitpunkt implementiert und unter einer Open-Source Lizenz freigegeben werden soll.

Zusammen mit der Abschlussarbeit wurde dem Themensteller ein Poster und eine Zusammenfassung per E-Mail übergeben.

Einverständnis mit der Veröffentlichung der folgenden Informationen im Internet

Anzeige des Namens:	Ja
Anzeige des Firmennamens:	Ja
Meine Mail-Adresse:	tjaehnel@jonmedia.net
Link auf Firmen-Homepage:	http://da.jonmedia.net

Unterschrift Student: _____

Tobias Jähnel

Erklärung

Ich erkläre hiermit, dass ich die Diplomarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Tobias Jähnel

Vorwort

Motivation

Bereits im Jahr 2002, als ich das erste Mal mit Windows XP in Berührung kam, fiel mir die Software „Windows Offlinefiles“ auf. Damit konnte ich Verzeichnisse, die auf meinem Netzwerkserver lagen als *Offline* kennzeichnen und Windows sorgte dafür, dass diese Dateien auch dann verfügbar waren, wenn ich nicht mit dem Netzwerk verbunden war. Da ich nach einiger Zeit gravierende Fehler und Unbequemlichkeiten dieser Software erkannte, setzte ich sie nur noch selten ein. Da ich neben Windows auch Linux sehr intensiv erprobte, suchte ich nach einer ähnlichen Software für diese Plattform, wurde aber nicht fündig.

In einer Unterhaltung zwischen Vorlesungsstunden im Sommer 2006 sprach Prof. Dr. Trommler genau dieses Thema an. Er hatte auf dem Linux-Kongress, der kurz vorher in der FH stattfand mit Volker Lendecke gesprochen. Dieser arbeitet am Samba Projekt [33] und vermisst ebenfalls eine Offlinefiles Lösung für Linux.

Aus dieser Notwendigkeit heraus entstand die Idee ein Offlinefilesystem für Linux zu entwickeln. Prof. Dr. Trommler und ich haben uns darauf geeinigt, dass ich in meiner Diplomarbeit bereits bestehende Lösungen betrachte und aus den Erkenntnissen ein Konzept für ein Offlinefilesystem erstelle. Im darauf folgenden Semester soll dieses Filesystem dann, im Rahmen meines Master of Science Studiums, implementiert wer-

den.

Hilfsmittel

Zur Umsetzung dieser Diplomarbeit wurde das Textsatzsystem TEXmit LATEXverwendet. Alle Abbildungen wurden mit Hilfe von OpenOffice.org[30] gezeichnet.

Aufbau

Die Diplomarbeit ist in drei Teile gegliedert. Der erste Teil befasst sich mit den Grundlagen von verteilten Dateisystemen und der Offlinefähigkeit. Ausserdem werden Anforderungen an Offline-Dateisysteme aufgestellt. Im zweiten Teil werden verschiedene Dateisysteme verglichen und diese aufgrund ihrer Offlinefähigkeit und Einsetzbarkeit bewertet.

Aus den Erfahrungen der verglichenen Dateisysteme wurden grundlegende Konzepte für ein eigenes, offlinefähiges Dateisystem entworfen. Diese werden im dritten Kapitel beschrieben. Die Implementierung wird zu einem späteren Zeitpunkt durchgeführt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1.

Einsatz von Offlinefilesystemen

1.1. Verteilte Dateisysteme

In nahezu allen Arbeitsgruppen werden mittlerweile PC's eingesetzt. Dies erfordert einen bequemen und reibungslosen Austausch von Daten zwischen den Mitarbeitern. Im Regelfall werden die gemeinsam genutzten Daten auf einem Dateiserver abgelegt. Die Mitarbeiter haben von ihren Arbeitsplatzrechner aus direkten Zugriff darauf. Zu diesem Zweck gibt es verschiedene Möglichkeiten wie beispielsweise FTP-Server oder Versionsverwaltungssysteme. In dieser Diplomarbeit wird die mittlerweile am weitesten verbreitete Technik betrachtet: „Verteilte Dateisysteme“, oder auch „Netzwerk-Dateisysteme“.

Die meisten Netzwerk-Dateisysteme arbeiten nach dem Client-Server Prinzip. Der Server legt fest, welche Verzeichnisse für Clients sichtbar sein dürfen (export) und die Client-Rechner binden diese lokal ein (mount). Der Einbinde-Vorgang ist der gleiche, der auch bei lokalen Dateisystemen ausgeführt wird. Unter Unix/Linux Betriebssystemen geschieht dies durch den Mountvorgang. Anders als bei einem lokalen Dateisystem (ext3, fat32 etc.) wird kein lokaler Datenträger, sondern das Dateisystem von einem an-

deren Rechner an einer beliebigen Stelle im Dateisystem eingebunden. Bei Windows hingegen wird das entfernte Dateisystem im Allgemeinen an einen freien Laufwerksbuchstaben gebunden. Dadurch macht es für den Benutzer keinen Unterschied, ob er auf der lokalen Festplatte oder einem Verzeichnis auf einem anderen Rechner arbeitet. Beim Abspeichern einer Datei wird diese direkt auf dem anderen Rechner abgelegt und steht somit sofort allen Benutzern zur Verfügung.

Der Bezechnung „verteiltes Dateisystem“ rührt daher, dass viele Netzwerk Dateisysteme mehr können als nur Verzeichnisse über das Netzwerk zu exportieren. So ist ein Verzeichnis beispielsweise nicht unbedingt komplett auf einem Rechner, sondern über mehrere Server *verteilt*. [19]

1.2. Funktionsweise eines Netzwerk Dateisystems

Um Transparenz zu erreichen, wird eine Abstraktionsebene eingeführt, die Dateizugriffe wie beispielsweise *open*, *close*, *read* oder *write* auf das entsprechende Dateisystem umsetzt. Der Benutzer sieht das zugrundeliegende Dateisystem nicht. Somit macht es keinen Unterschied, ob es sich um eine lokale Festplatte mit *ext3*, oder *fat32* Dateisystem handelt, oder es in Wirklichkeit auf einem anderen Rechner liegt.

Linux verwendet ein sogenanntes Virtual File-System (VFS). Dies ist eine Abstraktionsebene zwischen dem Anwendungsprogramm und dem Dateisystem (Abb. 1.1). Anwendungen stellen alle Anfragen an das VFS und dieses gibt es an den Treiber für das richtige Dateisystem weiter, welches die Befehle dann entsprechend umsetzt. Es muss lediglich für jedes verwendete Dateisystem ein VFS Modul zur Verfügung stehen. An dieser Stelle kann auch ein Treiber für ein verteiltes Dateisystem (in diesem Beispiel NFS), eingehängt werden.

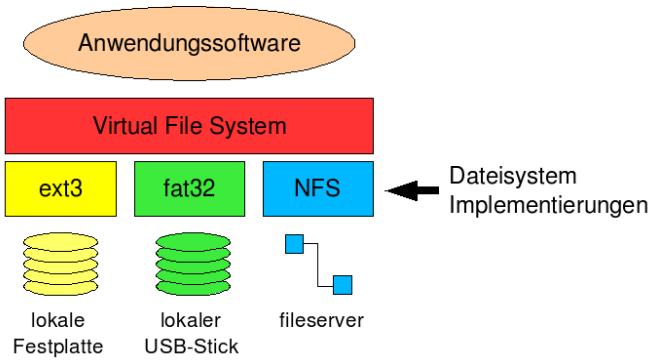

Abbildung 1.1.: Virtual File System Abstraktionsebene

1.3. Disconnected operation

Der große Nachteil eines verteilten Dateisystems ist, dass zur Bearbeitung der Dateien stets eine Netzwerkverbindung zum Server verfügbar sein muss. Dies ist allerdings nicht immer möglich, oder kann zu Problemen führen. Folgende Szenarien sind typisch:

- Bei der Arbeit an einem Dokument bricht die Verbindung zum Server unerwartet ab.
- Ein Mitarbeiter der Firma möchte Dokumente zuhause oder unterwegs bearbeiten, wo er keine Verbindung zum Firmennetzwerk hat.

1.3.1. Unerwarteter Verbindungsverlust

Bei verteilten Dateisystem führen Verbindungsprobleme in der Regel zu einem Abbruch. Antwortet der Server bei einem Schreibzugriff für eine gewisse Zeit nicht, so gibt es einen Timeout[1]. Dieser Fehler wird an den Benutzer gemeldet und die Operation abgebrochen.

[1] Um zu vermeiden, dass ein Prozess hängen bleibt, wenn der Server nicht antwortet, definiert man eine maximale Zeitspanne. Ist nach Ablauf dieser Wartezeit keine Antwort eingetroffen, geht das System von einem Serverausfall aus und bricht ab.

Disconnected Operation bedeutet, dass der Benutzer mit gemeinsamen Dateien weiterarbeiten kann, wenn die Netzwerkverbindung getrennt wurde. Die Dateien werden deshalb zusätzlich in einem Cache (Zwischenspeicher) auf dem Client abgelegt, auf dem das Betriebssystem während eines Netzwerkfehlers zurückgreifen kann.

1.3.2. Offlinefähigkeit

Die *Offlinefähigkeit* ist ein Spezialfall der *Disconnected Operation*. Möchte ein Mitarbeiter zuhause oder unterwegs an seinen Dokumenten weiterarbeiten, wird er, wenn kein Offlinefilesystem zur Verfügung steht, wie folgt vorgehen:

- Er kopiert die gewünschten Dateien von der gemeinsam genutzten Netzwerkfreigabe auf die Festplatte seines Notebooks.
- Kehrt er zurück in die Firma, kopiert er die geänderten Dateien wieder zurück.

Dies ist eine Art *manuelle disconnected Operation*. Dabei können einige Probleme auftreten. Z.B. kann es passieren, dass er Änderungen, die in der Zwischenzeit von anderen Benutzern gemacht wurden, überschreibt. Ausserdem muss er sich merken, welche Dateien er verändert hat um unnötiges Kopieren zu vermeiden.

Ein offlinefähiges verteiltes Dateisystem, kurz „Offlinefilesystem“, sorgt selbständig dafür, dass Dateien offline verfügbar sind und Änderungen sauber zurück auf den Server geschrieben werden. Das Problem dabei ist jedoch, dass nun wieder mehrere Kopien der gleichen Dateien existieren, die unabhängig voneinander geändert werden können. Also genau das, was man durch den Einsatz von verteilten Dateisystemen verhindern möchte. Die Hauptaufgabe eines offline Dateisystems besteht also darin, die Dateien auf Server und Client so konsistent wie möglich zu halten.

1.4. Kriterien zum Vergleich von Offlinefilesystemen

Im folgenden werden Anforderungen aufgelistet, denen ein Offlinefilesystem für den Produktiveinsatz genügen sollte. Anhand dieser werden im Anschluss verschiedene Dateisysteme verglichen.

Netzwerk Transparenz Wie bei verteilten Dateisystemen üblich, darf es für den Client keinen Unterschied machen, ob er auf einem lokalen oder einem entfernten Dateisystem arbeitet.

Offline Transparenz Es darf für den Client keinen Unterschied machen ob eine Netzwerkverbindung besteht oder nicht.

Automatische Synchronisation Bei Anbindung an das Netzwerk, müssen die veränderten Daten auf Client und Server selbständig auf den gleichen Stand gebracht werden.

In-Time Synchronisation Bei unvorhergesehenem Verlust der Netzwerkverbindung muss der Benutzer so weiterarbeiten können wie gehabt.

Freiheit von Inkonsistenzen Fällt die Verbindung zum Server unvorhergesehen aus, muss der Benutzer seine Arbeit an geöffneten Dateien beenden können.

Replikation Um Ausfälle so gering wie möglich zu halten, sollte es möglich sein, mehrere Server zu verwenden, die Daten redundant halten.

Skalierbarkeit Die Erweiterung von Speicherplatz sollte ohne Ausfälle geschehen.

Benutzerfreundlichkeit Der Benutzer muss eine komfortable Möglichkeit haben, die Dateien auszuwählen, die er offline verfügbar haben möchte. Ausserdem müssen Konflikte gemeldet werden und der Benutzer muss eine einfache Möglichkeit haben, diese zu beheben.

Einbindung in die vorhandene Infrastruktur Das Dateisystem muss sich an die bestehende Infrastruktur anpassen, nicht umgekehrt. Anpassungfähigkeit an verschiedene Umgebungen auch bei Aufbau einer neuen Infrastruktur erforderlich, denn die Entscheidung für ein System wird i.d.R. nicht aufgrund des Dateisystemes getroffen.

Kapitel 2.

Analyse verteilter Dateisysteme

2.1. Datenabgleich mit Unison

Unison[41] ist kein verteiltes Dateisystem, sondern ein Programm, das zum Abgleich von Dateien dient. Es soll hier trotzdem kurz vorgestellt werden, da es dem Benutzer die Möglichkeit gibt offline zu arbeiten, indem es lokale und entfernte Verzeichnisse synchronisiert. Es sind viele kostenpflichtige und kostenlose Lösungen verfügbar, die nach dem gleichen Konzept arbeiten. Da Unison als Open-Source unter der General Public License (GPL)[16] freigegeben und zudem weit verbreitet ist, dient dies hier als Beispiel. Eine weitere Software ist *SyncToy* von Microsoft, die in Abschnitt 2.4.8 vorgestellt wird.

2.1.1. Funktionsweise

Der Workflow bei Anwendung einer Synchronisations-Software ist typischerweiser wie folgt:

- Der Benutzer des mobilen Rechners wählt die Verzeichnisse aus, die er unterwegs

verfügbar haben möchte und ein lokales Verzeichnis in das die Dateien kopiert werden sollen. Dies teilt er dem Programm Unison mit.

- Die Software kopiert nun die gewählten Dateien vom Netzwerk in das lokale Verzeichnis. Der Benutzer kann diese nun unterwegs bearbeiten.
- Kommt er zurück in die Firma und der Netzwerkzugang ist wieder verfügbar, startet er Unison erneut und lässt das lokale und das entfernte Verzeichnis abgleichen. Die lokal veränderten Dateien werden auf den Server kopiert und umgekehrt. Wurde eine Datei sowohl lokal als auch entfernt verändert, liegt ein Konflikt vor, der dem Benutzer gemeldet wird. In diesem Fall muss dieser selbst entscheiden was zu tun ist.

Mit Hilfe von Synchronisationstools können zwei beliebige lokale Verzeichnisse synchronisiert werden. Dadurch ist es beispielsweise auch möglich, Daten auf einem USB-Stick mitzunehmen. Möchte der Benutzer Dateien zwischen der lokalen Festplatte und einem Netzlaufwerk synchronisieren, muss die Freigabe dazu lokal eingebunden sein. Hierzu ist zusätzlich ein verteiltes Dateisystem notwendig (Kapitel 1).

Derartige Tools haben allerdings einige Nachteile gegenüber Offline-Dateisystemen. Der Benutzer ist für die Sychronisation selbst verantwortlich. Er bestimmt, wann die Daten abgegelichen werden. Dies ist eine potentielle Fehlerquelle, da dieser Abgleich schell einmal vergessen wird. Würde man die Synchronisation automatisieren, müsste dies laufend geschehen, damit sichergestellt ist, dass die Daten beim Entfernen der Netzwerkverbindung aktuell sind. Dies ist sehr ineffizient, da bei jedem Abgleich sämtliche Dateien auf Veränderung überprüft werden. Ausserdem ist keine Offline-Transparenz vorhanden. Ist der Rechner mit dem Firmennetzwerk verbunden, sieht der Anwender immer zwei Verzeichnisse: das lokale und das auf dem Dateiserver. Dies ist verwirrend und ausserem ebenfalls eine Fehlerquelle.

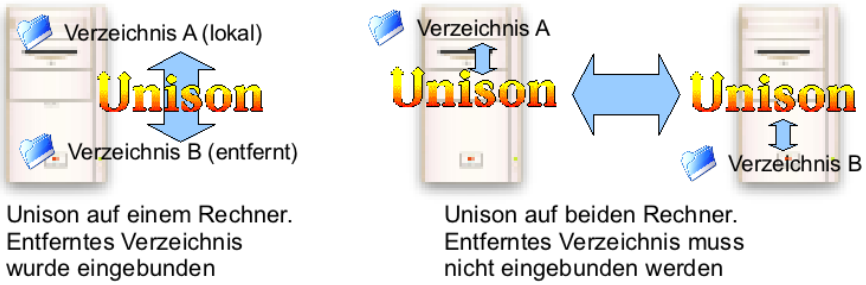

Abbildung 2.1.: Nutzung des rsync Algorithmus.

2.1.2. rsync Algorithmus

Zum Abgleich von Daten in Unison dient der rsync-Algorithmus. Eigentlich ist dieser nicht darauf ausgelegt, zwei lokale Verzeichnisse zu synchronisieren. Er hat vielmehr seine Stärken im Abgleich von Dateien über eine langsame Netzwerkverbindung. Dazu laufen auf beiden Rechnern rsync-Prozesse, welche per Secure Shell (SSH) Protokoll miteinander kommunizieren. Über den Austausch von Prüfsummen[1] bestimmen sie, welche Teile einer Datei unverändert sind und übertragen im Anschluss nur die Änderungen. [40]

Natürlich ist es auch möglich, zwei lokale Verzeichnisse miteinander zu synchronisieren. Ist das entfernte Dateisystem in den lokalen Verzeichnisbaum eingehängt, kann so eine Synchronisierung geschehen, ohne auf dem Server eine Software zu installieren. In diesem Szenario ist allerdings zu beachten, dass die Netzwerkeffizienz vollkommen verloren geht. (Abb.: 2.1)

[1] Mit Hilfe von Prüfsummen wird normalerweise die Integrität bei Datenübertragung sichergestellt. Kommt der Empfänger bei der Berechnung zu einem anderen Ergebnis als der Sender, gab es einen Übertragungsfehler [52]. Was der rsync Algorithmus hier einsetzt ist eher als Hash zu bezeichnen [47].

2.1.3. Einsatzmöglichkeiten

Wie eingangs erwähnt, ist Unison lediglich eine Software zum Synchronisieren von Dateien. Allerdings bringt es mit dem rsync Algorithmus einen guten Lösungsansatz für eines der größten Probleme von Offlinefilesystemen: das Aufspüren von Änderungen im Dateisystem und den effizienten Abgleich. Wer seine Daten lediglich unterwegs verfügbar haben möchte und dabei auf Transparenz und Automatisierung verzichten kann, wird mit Unison gut arbeiten können. Ausserdem gibt es auch eine mehr oder weniger komfortable grafische Benutzeroberfläche dafür.

2.2. NFS - Network File System

2.2.1. Geschichte

Das Network File System (NFS) ist ein klassisches Netzwerkdateisystem für Unix. Ursprünglich wurde das Protokoll von Sun entwickelt. Ziel war es, einen transparenten Zugriff auf Dateien eines Servers zu ermöglichen. Dabei wagte Sun einen Schritt, der im Jahr 1984 noch unüblich war. Sie verzichteten auf die Geheimhaltung und gaben die Spezifikation des NFS kostenlos heraus. Nicht zuletzt diese Tatsache machte das Dateisystem so populär.

Mit Version 3 gab Sun die Spezifikation und Weiterentwicklung an die Internet Engineering Task Force (IETF) ab. Dies ist eine online community, die sich mit dem Vorantreiben von Internet Technologien beschäftigt. Weitere Informationen sind auf deren Website unter [21] zu finden. NFS ist unter der GPL[16] veröffentlicht worden.

2.2.2. Konzept

Das NFS ist ein reines Client-Server Protokoll. Die Server exportieren ihre Dateisysteme und die Clients binden diese ein. Alle Server arbeiten unabhängig von einander, weshalb die Clients genau wissen müssen, welche Freigaben von welchen Servern eingebunden werden. Eine Server-Replikation ist mit „Bordmitteln“ nicht möglich. Die Einrichtung von NFS ist nicht sehr aufwändig. Auf dem Server werden die zu exportierenden Verzeichnisse in der Datei */etc/exports* definiert. Auf den Clients werden anschließend die gewünschten Verzeichnisse eingebunden.

2.2.3. Zugriffssteuerung

Der Server legt für jedes exportierte Verzeichnis separat fest, welche Clients dieses einbinden dürfen. Dieser Zugriff kann durch weitere Parameter spezifiziert werden. Dazu gehört beispielsweise, ob nur gelesen oder auch geschrieben werden darf.

Da NFS ursprünglich nur unter UNIX verwendet wurde, sind die Datei-Zugriffsrechte auf dieses abgestimmt. Jeder Benutzer, sowie jede Gruppe besitzt eine jeweils eindeutige Nummer. Um sich diese Zahlen nicht merken zu müssen, werden diese in den Dateien */etc/passwd* bzw. */etc/group* eindeutigen Namen zugeordnet. (z.B. kmeier, fschmidt, buchhaltung). Wie bei den üblichen Unix-Dateisystemen (z.B. ext3) kennt auch NFS nur diese Identifikations-Zahl. Es ist daher erforderlich, dass auf allen Rechnern die Zuordnung zu den Namen identisch ist.

Aus diesem Grund wird NFS oft zusammen mit Network Information Service (NIS) (früher YP - Yellow Pages) verwendet. Bei diesem System handelt es sich um einen Verzeichnisdienst, der auf dem NIS-Server gespeicherte Benutzer und Gruppen an die Clients exportiert. Auf diese Weise wird sichergestellt, dass die Benutzernummern, die mit dem NFS übertragen werden, auch zu den korrekten Benutzernamen aufgelöst wer-

den können. Heutzutage werden allerdings meistens andere Systeme, wie beispielsweise LDAP, eingesetzt.

2.2.4. Technische Details von NFSv3

Die ursprüngliche, von Sun veröffentlichte NFS-Version ist NFSv2. Sie unterstützt die Übertragung nur per User Datagram Protocol (UDP). NFSv3 unterstützt sowohl das UDP als auch das Transmission Control Protocol (TCP). In beiden Fällen ist das NFS-Protokoll zustandslos. Das heißt, es merkt sich nicht den Zustand der verbundenen Clients, wodurch sämtliche Anfragen unabhängig sind. Da das TCP ein verbindungsorientiertes Protokoll ist, kümmert sich die Implementierung selbst darum, dass die Pakete ankommen. Beim UDP muss das der NFS Client bzw. Server erledigen.

2.2.5. Caching und Offlinefähigkeit

Das NFS besitzt seit Version 3 einen Cache. Dieser dient dazu, die Geschwindigkeit bei mehreren Zugriffen auf die gleiche Datei zu erhöhen. Wie bereits erwähnt, ist das NFS zustands- und verbindungslos. Falls der Server, z.B. aufgrund eines Neustarts, für kurze Zeit nicht erreichbar ist, wiederholt der Client seine Anfrage, bis er eine Antwort bekommt. Der Zugriff wird also für die Zeit der Abwesenheit des Servers blockiert und es gehen keine Daten verloren. Das gleiche geschieht bei kurzzeitigem Verlust der Netzwerkverbindung. Im Idealfall bemerkt der Client diesen Ausfall garnicht.

Von disconnected operation kann allerdings in diesem Zusammenhang nicht gesprochen werden. Es wird lediglich ein kurzzeitiger Ausfall der Verbindung überbrückt. Um Daten lesen und schreiben zu können, muss in jedem Fall eine Verbindung bestehen.

2.3. NFSv4 - Network File System Version 4

Version 4 des Network File Systems befindet sich noch in der Entwicklung. Es sind bisher nur Prototypen und keine stabile Version verügbar.

Der Fokus der Neuerungen wird stark auf folgende Punkte gelegt:

- Performance
- Sicherheit
- Unterstützung verschiedener Plattformen
- Erweiterbarkeit

Auch in Version 4 des NFS wird Offlinefähigkeit vernachlässigt. In den Überlegungen zum Design (siehe RFC 2624[37]) ist in Abschnitt 4.3 folgendes über dieses Thema zu lesen:

> While very desirable, disconnected operation has the potential to inflict itself upon the NFS protocol in an undesirable way as compared to traditional client caching. Given the complexities of disconnected client operation and subsequent resolution of client data modification through various playback or data selection mechanisms, disconnected operation should not be a requirement for the NFS effort.

In RFC 3530[38], wo das NFSv4 Protokoll spezifiziert wird, gibt es keine Angabe über disconnected Operation.

2.4. SMB/CIFS und Windows Offlinefiles

Die Netzwerkdateisystem-Lösung von Microsoft besteht aus drei Bestandteilen. Datenfreigabe und -zugriff geschieht über das Common Internet File System (CIFS) Protokoll.

Das Microsoft Distributed File System (DFS) setzt die Freigaben von verschiedenen Dateiservern zu einem globalen Namensraum zusammen. Auf den Clients ist die Software *Windows Offlinefiles* installiert, welche das System um Offlinefähigkeit erweitert. Diese drei Komponenten können auch unabhängig voneinander eingesetzt werden.

2.4.1. SMB/CIFS Protokoll

Das Server Message Block (SMB) Protokoll wurde ursprünglich von IBM als Netzwerkdateisystem für DOS entwickelt. Aus diesem Grund war es auch sehr stark auf DOS abgestimmt. Microsoft hat dieses System bald als Standard für seine Produkte verwendet und entwickelt es seitdem stetig weiter, um es an die aktuellen Windows Versionen anzupassen. In Windows Vista wurde SMB 2.0 eingeführt. Die bekannteste SMB Implementierung für nicht-Windows Betriebssysteme ist Samba [33], welches unter der GPLv3[18] freigegeben wurde.

SMB basiert auf NetBIOS, einer Abstraktionsschicht zum Zugriff auf das Netzwerk. Seit Windows 2000 und Samba 3 ist es jedoch auch möglich SMB direkt über TCP zu betreiben. Im Jahr 1996 benannte Microsoft SMB in CIFS um. Dies scheint allerdings nicht konsequent durchgezogen worden sein. In der Praxis wird SMB und CIFS meist synonym verwendet.

Wird SMB separat eingesetzt, funktioniert es im Wesentlichen wie NFS (Siehe Abschnitt 2.2). Die Server exportieren Verzeichnisbäume über das Netzwerk und die Clients binden diese nach Belieben ein. In Heterogenen Systemen mit Windows, Unix und Mac Rechnern wird oft auf SMB zurückgegeriffen, da es auf allen Systemen sehr gut unterstützt wird.

Die Verwaltung der Zugriffsrechte ist in aktuellen SMB Versionen so ausgelegt, dass sie mit den komplexen Windows Access Control List (ACL)[43] zurechtkommt. Es ist jedoch möglich, diese zu vereinfachen und dadurch auf Unix Zugriffsverwaltung umzu-

setzen. Ein weiteres Problem bei der Interoperabilität ist, dass Windows im Gegensatz zu Unix nicht zwischen Groß- und Kleinschreibung unterscheidet. Die Entwickler der Unix Portierung von SMB haben dies dadurch gelöst, dass sie entweder alles in Kleinbuchstaben umwandeln oder alle möglichen Kombinationen von Groß- und Kleinbuchstaben durchprobieren.

2.4.2. DFS

Das Microsoft DFS ist Bestandteil von Windows Server ab Version 2000. DFS ermöglicht es, einen globalen Namensraum aufzubauen. Ein Server exportiert die Wurzel des Verzeichnisbaumes (DFS Root). Die Freigaben von anderen Servern werden an beliebiger Stelle in diesen Verzeichnisbaum eingehängt. Dadurch ist für den Benutzer nicht sichbar, auf welchem physikalischen Server sich die Daten befinden. Weitere Informationen zum globalen Namensraum gibt es in Abschnitt 2.5.2. Eine gelungene Demonstration der Funktion von DFS ist auf der Microsoft Website unter [26] zu finden. Mittlerweile unterstützt auch Samba, die Open-Source SMB Implementierung DFS.

2.4.3. Offlinefiles

Microsoft hat in Windows 2000 die sogenannten *Windows Offlinefiles* eingeführt. Dieses Feature ist auch unter Windows XP Professional und allen Windows Vista Editionen ausser Starter, Home Basic und Home Premium vorhanden. Windows Offlinefiles ist eine Software, die auf dem Client installiert ist. Sie kopiert den Inhalt von Netzwerkfreigaben oder Ordnern auf anderen Rechnern in den lokalen Client Side Cache (CSC). Dadurch sind diese Dateien auch vorhanden, wenn keine Verbindung zum Netzwerk besteht.

Der Benutzer des Clients navigiert mit dem Windows Explorer über die Netzwerkumge-

bung oder ein eingebundenes Netzlaufwerk zu dem Ordner, den er unterwegs verwenden möchte. Über das Kontextmenü markiert er diesen als „Offline verfügbar“. Windows kopiert den Ordner daraufhin in den CSC auf der lokalen Festplatte. Wird die Verbindung zum Server später getrennt, kann der Benutzer auf diese Dateien weiterhin zugreifen. Der Zugriff ist transparent. D.h. der Pfad zur Datei ist immer der selbe, unabhängig davon, ob eine Verbindung zum Server besteht oder nicht. Offlinefiles leitet alle Operationen bei bestehender Verbindung an den Server und bei getrennter Verbindung an den CSC.

Die Dokumentation zu den Offlinefiles, die Microsoft zur Verfügung stellt, ist sehr dürftig. Es gibt einige Anleitungen, die beschreiben, wie die Offlinefiles verwendet werden, über die zugrundeliegenden Konzepte existiert allerdings kaum Material.

2.4.4. Quick und Full Sync

Windows Offlinefiles unterscheidet zwischen drei Kategorien von Dateien:

Permanante (Pinned) Offline files Diese Dateien wurden vom Benutzer als „Offline verfügbar“ gekennzeichnet, da er sie unterwegs ansehen oder bearbeiten möchte.

Sparsely Cached files Diese Dateien wurden auf dem Server neu angelegt, sind aber vom Client noch nicht in den CSC übertragen worden.

Temporary Cached files Diese Dateien werden ebenfalls im CSC abgelegt, allerdings nur temporär, um Ausfälle zu überbrücken und die Zugriffsgeschwindigkeit zu erhöhen.

Ein Spezialfall sind die *Temporary Cached files*. Öffnet der Benutzer eine Datei auf einem Netzlaufwerk, kopiert Windows diese gleichzeitig in den CSC. Dies geschieht auch dann, wenn die betreffende Datei nicht als *Offline verfügbar* gekennzeichnet wur-

de. Bricht die Netzwerkverbindung ab, während der Benutzer die Datei noch bearbeitet, aktualisiert Windows beim Abspeichern nur die Kopie im CSC. Beim nächsten Synchronisationsvorgang wird diese dann auf den Server kopiert. Der Speicherplatz, der für Temporary Cached files in Anspruch genommen werden darf, kann vom Benutzer begrenzt werden. Beim Überschreiten dieser Grenze werden ältere Dateien gelöscht.

Windows führt im Leerlauf von Zeit zu Zeit eine automatische Synchronisation der Dateien durch. Bei diesem *Quick Sync*, der darauf ausgelegt ist, schnell zu sein, wird lediglich für Vollständigkeit, nicht jedoch für Aktualität gesorgt. Alle Sparsely Cached Files werden in den CSC übernommen, damit sichergestellt ist, dass alle Dateien offline vorhanden sind. Die Aktualität der anderen Dateien wird nicht überprüft. Wie oft dieser Quick-Sync durchgeführt werden soll, kann vom Benutzer konfiguriert werden.

Der *Full Sync* wird, je nach Konfiguration, bei der Anmeldung bzw. Abmeldung ausgeführt. In diesem Fall durchsucht Windows sämtliche Dateien und bringt diese auf den aktuellen Stand. Da dies einige Zeit in Anspruch nimmt, wird der Anmelde- bzw. Abmeldevorgang entsprechend gebremst. [14]

2.4.5. Auflösung von Konflikten

Es ist nirgends dokumentiert, wie *Windows Offline-Files* einen Konflikt erkennt. Ein kurzer Test lässt allerdings darauf schließen, dass lediglich ein Vergleich der Erstellungs- bzw. Modifikationszeit von Dateien durchgeführt wird.

Vermutlich läuft die Erkennung von Konflikten wie folgt ab:

- Windows merkt sich den Zeitpunkt der letzten Synchronisation und erkennt durch Vergleich dieser Zeit mit der Modifikationszeit, ob eine bestimmte Datei lokal oder entfernt verändert wurde.
- Ist eine Datei nur auf einer Seite vorhanden, so kann durch Vergleich der Erstel-

lungszeit der Datei und der letzten Synchronisationszeit festgestellt werden, ob sie gelöscht oder neu angelegt wurde.

- Hat der Benutzer eine Datei umbenannt, so wird beim Synchronisieren die alte Datei gelöscht und die neue angelegt.

Um komplizierteren Konfliktsituationen aus dem Weg zu gehen, verbietet Microsoft das Umbenennen von Ordnern, wenn keine Verbindung zum Server besteht.

2.4.6. Bewertung

Da die Software Windows Offline Files mit Windows mitgeliefert wird, kann sie einfach über das Windows Setup installiert werden und integriert sich nahtlos in die Umgebung. Die Software ist einfach zu bedienen und funktioniert auf jeder SMB/CIFS Netzwerkfreigabe. Allerdings arbeitet es nur mit SMB/CIFS Freigaben zusammen. Unter [29] wird beschrieben, dass Versuche, Offlinefiles mit Andrew File-System (AFS) zu verwenden, Probleme machen. AFS meldet nicht, wenn die Verbindung getrennt wurde.

Leider kann man sich nur dann sicher sein, dass alle Daten aktuell sind, wenn man sich vom System abmeldet oder es herunterfährt. Dadurch wird der Ruhezustand (Hibernate), ein sehr praktisches Feature von Windows, unbrauchbar. Auch durch eine manuell initiierterte Syncronisation wird nur ein teilweiser *full sync* durchgeführt [14].

Der *full sync* durchsucht sämtliche Dateien auf Änderungen. Deshalb dauert der Anmelde- bzw. Abmeldevorgang bei großen Datenmengen sehr lange. Man sollte vor dem Verlassen des Büros also genug Zeit für die Synchronisation einplanen. Erfahrungen haben gezeigt, dass der Abgleich bei einer Datenmenge von 1-2GB so langsam wird, dass es nicht mehr genutzt werden kann. Ausserdem benötigt es bereits einige Zeit, um die Synchronisation zu starten, weshalb auch bei wenigen Dateien deutlich merkbare Wartezeiten entstehen.

Übergang zwischen Online und Offline

Eigene Erfahrungen mit den Offline Files unter Windows XP zeigten, dass die Online/Offline Erkennung noch sehr fehlerhaft ist. Obwohl keine Netzwerkverbindung besteht, öffnet sich zeitweise das Syncronisationsfenster, um nach einiger Wartezeit zu melden, dass eine Syncronisation nicht möglich ist. Umgekehrt erkennt Windows beim Anstecken der Netzwerkverbindung zwar, dass eine Syncronisation durchgeführt werden müsste, bricht dies manchmal allerdings mit der Meldung ab, dass der Server nicht verfügbar ist. Dies tritt vor allem dann ein, wenn die Verbindung nicht sofort nach dem Einstecken des Netzwerkkabels verfügbar ist, weil die Netzwerk-Einstellungen, die über Dynamic Host Configuration Protocol (DHCP)[2] bezogen werden, noch nicht vorliegen.

Sehr unschön ist auch, dass Offlinefiles offensichtlich davon ausgeht, dass bei einer bestehenden Netzwerkverbindung zwingend sofort alle Server verfügbar sind. Arbeitet man abwechselnd in verschiedenen Netzwerken, wird man bei jedem Synchronisationsvorgang mit einem Fenster darauf hingewiesen, dass nicht alles abgegelichen werden konnte. Dieses muss man zeitweise per Hand schließen um weiterarbeiten oder den PC herunterfahren zu können.

Navjot Virk, der Program Manager der Offline Files in Windows Vista spricht in seinem Weblog [42] ausserdem davon, dass unter Windows XP alle Programme geschlossen sein müssen, um die Synchronisation auszuführen. Einige dieser Probleme wurden von Microsoft erkannt und sollen in Windows Vista beseitigt sein (siehe 2.4.7).

[2] „Das DHCP ermöglicht mit Hilfe eines entsprechenden Servers die dynamische Zuweisung einer IP-Adresse und weiterer Konfigurationsparameter an Computer in einem Netzwerk.“[46]

2.4.7. Windows Vista und Offline Files

Die Editionen *Enterprise*, *Business* und *Ultimate* aus der Windows Vista Produktreihe enthalten die Windows Offline Files. Microsoft hat die Implementierung komplett überarbeitet und in vielen Punkten verbessert. Es gibt einige Neuerungen, die die Verwendung von Offlien-Files noch komfortabler machen soll. So ist es für den Benutzer nun beispielsweise möglich, in den Offline-Modus zu wechseln, ohne die Netzwerkverbindung zu trennen.

Seamless Transition

Unter Windows XP traten beim Wechsel zwischen Online und Offline oft Probleme auf, bei denen der Benutzer eingreifen musste. Durch ein Feature, das Microsoft *Seamless Transition* nennt, wurde dieser Vorgang nun verbessert. [42] und [24] informieren über Details.

Schnellere Synchronisation

Ausserdem informiert Navjot Virk in seinem Weblog [42] darüber, dass nun ein schnellerer Algorithmus verwendet wird um Unterschiede zwischen Dateien zu erkennen. Nähere Informationen über dieses Algorithmus gibt er allerdings nicht. Er erklärt lediglich eine Funktion namens *Bitmap Differential Transfer*. Hierbei wird nicht die gesamte Datei, sondern nur die Änderungen vom Client zum Server übertragen. In der Richtung vom Server zum Client funktioniert dies allerdings nicht.

Eine Beschreibung aller Neuerungen der Offline Files unter Windows Vista ist unter [24] zu finden.

2.4.8. Exkurs: SyncToy

Das Thema Synchronisation wird bei Microsoft in der letzten Zeit sehr hochgehoben. In Windows Vista wurde der Sync-Center eingeführt, um für alle Synchronisationvorgänge eine gemeinsame Benutzeroberfläche zu haben. Dazu gehören beispielsweise *Active Sync* zum Abgleich von Windows Mobile Geräten[3] und auch *Windows Offlinefiles*. Ausserdem gibt es seit längerer Zeit SyncToy, welches nach dem gleichen Prinzip wie Unison (Abschnitt 2.1) arbeitet. Es sorgt dafür, dass zwei beliebige Verzeichnisse per Knopfdruck auf den gleichen Stand gebracht werden, bietet allerdings noch wesentlich mehr Funktionen. SyncToy besitzt auch eine bequemere Oberfläche und ist besser in Windows integriert. Im Gegensatz zu Unison arbeitet es allerdings nur auf Windows Plattformen. Da auch SyncToy kostenlos ist, sollte man dieses Tool in der Regel Unison vorziehen.

2.5. AFS - Andrew File System

2.5.1. Geschichte

AFS wurde im Jahr 1984 an der Carnegie Mellon University (CMU) in Kooperation mit IBM entwickelt. Es sollte, zusammen mit anderen Hardware- und Softwareprojekten, dafür sorgen, dass die Computer an der CMU ein einheitliches Gesicht bekommen. Im Jahr 1988 wurde eine Firma namens Transarc gegründet, die das Dateisystem kommerziell vertrieb. IBM kaufte diese Firma im Jahr 1999 auf und kurz darauf wurde der Vertrieb eingestellt.

Die Open-Source Community entwickelte das Dateisystem unter dem Namen *OpenAFS*

[3] Windows Mobile ist ein sehr kleines Betriebssystem für Mobiltelefone und PDAs (Personal Digital Assistent).

weiter und pflegt dieses bis heute. Es ist mittlerweile sehr stabil und läuft auf gängigen Betriebssystemen wie Linux, Windows und Mac OS X [15]. OpenAFS steht unter der Open-Source Lizenz „IBM Public License“[31].

2.5.2. Globaler Namensraum

Das Konzept der verteilten Dateisysteme NFS und SMB basiert auf Freigaben. Die Server stellen unabhängig voneinander über sogenannte „Freigaben“ oder „Exports“ bestimmte Verzeichnisbäume zur Verfügung, die jeder Client einzeln einbinden kann.

AFS ist prinzipiell dafür ausgelegt, von jedem Ort aus dem Internet verfügbar zu sein. Aus diesem Grund gibt es einen globalen Namensraum (Verzeichnisbaum) über den theoretisch sämtliche Freigaben der ganzen Welt verfügbar sind. Der Zugriff erfolgt über einen einzigen Mountpoint. Bei Linux ist das */afs*, bei Windows das Netzlaufwerk *afs*. Bei Zugriff auf eine Datei oder ein Verzeichnis in diesem Namensraum wird automatisch ermittelt, von welchem Server die Daten bezogen werden können. Dazu muss der AFS Client lediglich einen einzigen Server kennen, der ihm weitere Informationen über die Struktur des Namensraumes verschaffen kann. [15]

Das „Verzeichnis“ auf der ersten Ebene nach */afs* beschreibt den Namen der Zelle. Zellen sind Abschnitte, die komplett unabhängig administriert werden können. Jede Zelle besitzt ihre eigenen Clients, Server und Benutzer-Accounts. [28]

Die weitere Struktur des Namensraumes der Zelle wird vom Zellen-Administrator verwaltet. Sie besteht aus sogenannten Volumes, die in den Verzeichnisbaum an beliebiger Stelle eingehängt werden. Volumes sind die kleinste administrative Einheit. Sie sind in sich abgeschlossen und enthalten in der Regel Daten, die logisch zusammengehören. Volumes sind mit Unix Dateisystemen vergleichbar. Ein Volume besitzt eine Wurzel und weitere Volumes können an beliebigen Stellen eingehängt werden. Typischerweise ist ein Volume viel kleiner als eine Partition, aber viel größer als ein einzelnes Verzeichnis.

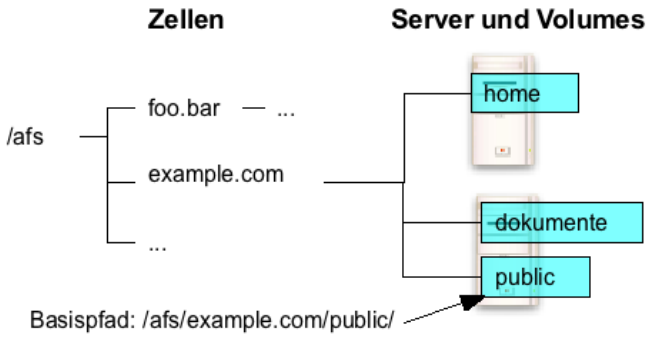

Abbildung 2.2.: Aufbau des AFS Namensraumes

[35] (Abb.: 2.2)

Dieses Konzept hat den Vorteil, dass alle Clients gleich konfiguriert werden können und alle Benutzer stets den gleichen Verzeichnisbaum sehen, unabhängig davon auf welchem Rechner sie angemeldet sind. Ausserdem ist es einfach, neue Freigaben und Server hinzuzufügen oder zu entfernen. Es ist also eine gute Erweiterbarkeit und Skalierbarkeit gegeben. Unter [26] ist eine Flash Animation zu finden, die das Konzept des globalen Namensraumes und die daraus entstehenden Vorteile von Microsoft DFS zeigt. Bei AFS funktioniert dies ähnlich.

2.5.3. Caching

Der AFS Client besitzt einen lokalen Cache um die Performance zu optimieren. Nur beim ersten Zugriff auf eine Datei wird diese vom Server geladen. Bei mehrfachem Zugriff auf die gleiche Datei muss diese nicht erneut übertragen werden, was sich vor allem bei einer langsamen Verbindung zum Server bemerkbar macht. Ändert sich der Inhalt einer Datei auf dem Server, informiert dieser automatisch alle Clients, bei denen diese Datei im lokalen Cache abgelegt ist. [28]

2.5.4. Bewertung

Während sich das ursprüngliche AFS nie durchgesetzt hat, ist openAFS, nicht zuletzt wegen der freien Verfügbarkeit, zu einem populären Dateisystem geworden. Leider trägt die Lizenz immer noch Altlasten mit sich, sodass es nicht Teil des offiziellen Kernels wird.

Bedauerlich ist, dass AFS immer noch disconnected Operation fehlt. Im Concurrent Versions System (CVS) repository befindet sich ein Zweig mit dem Namen „disconnected“, in dem eine „disconnected Operation Erweiterung“ für AFS entwickelt werden soll. Allerdings hat sich hier seit 2 Jahren nichts mehr getan. Das Projekt Coda (Siehe 2.6), ist unter anderem aus diesem Mangel heraus entstanden.

2.6. Coda Filesystem

Das Coda Dateisystem wurde 1987 unter der Leitung von M. Satynarayanan an der CMU in Pittsburgh (Pennsylvania) ins Leben gerufen. Ziel war es, das verteilte Dateisystem AFS zu verbessern. Da an der CMU bereits eine AFS Infrastruktur vorhanden war, kannten die Entwickler einige Probleme, welche sie durch Coda beheben wollten. Besonderer Wert wurde dabei auf hohe Verfügbarkeit gelegt, da AFS nicht sehr resistent gegen Ausfälle ist. M. Satynarayanan sagt in [34], dass es unvermeidlich ist, dass es Netzwerkausfälle gibt. Da dies bei den Benutzern an der CMU des öfteren zu Unmut führte, hat er sich entschlossen, mit Coda an diesem Schwachpunkt anzusetzten.

Bei den ersten Entwicklungen wurde die in AFS vorhandene Server-Replikation verbessert. Dies bedeutet, dass es mehrere Server gibt, die die gleichen Daten halten und sich selbständig syncronisieren. Somit steht bei Ausfall eines Rechners stets eine weitere Replikation zur Verfügung. In der weiteren Arbeit haben die Entwickler auch kurzzeitige Ausfälle des Netzwerkes berücksichtigt, indem sie einen intelligenten Cache-

Mechanismus für die Client-Seite entwarfen. Dadurch ist es möglich, kurzzeitig ohne Verbindung zum Server zu arbeiten (*Disconnected Operation*). Aufgrund der immer weiteren Verbreitung von mobilen PCs wurde die Idee geboren, dass auch längere Zeit ohne Verbindung zum Dateiserver mit den gemeinsam genutzten Daten gearbeitet werden könnte. Der Cache Mechanismus wurde daraufhin erweitert. [34]

In der Musik wird der Abschluss, der ein Stück sozusagen abrundet, als Coda (ital.: Schwanz) bezeichnet. Dies ist der Ursprung für den Namen dieses Dateisystems. Es soll das AFS sozusagen noch abrunden. Später wurde es von manchen als Abkürzung für „COnstant Data Availability“ bezeichnet. [34]

Das Coda Filesystem wird noch immer unter Leitung von M. Satyanarayana weiterentwickelt. Die wichtigsten Features sind im folgenden aufgelistet [13]:

- Disconnected Operation
- High Performance
- Server Replikation
- Security
- Resistenz gegen teilweise Netzwerkausfälle
- Bandbreiten-Anpassung
- Gute Skalierbarkeit

Da Coda komplett neu entwickelt wurde, ist es nicht an die Lizenz von AFS gebunden. Es ist unter der GPL[16], einige Teile auch unter der Lesser General Public License (LGPL)[17], freigegeben und Bestandteil aktueller Linux-Kernels.

2.6.1. Globaler Namensraum

Coda ist ein echtes verteiltes Dateisystem. Das heißt, es gibt mehrere Datei-Server im Netzwerk, die das Dateisystem zur Verfügung stellen. Der Client wendet sich nicht an einen bestimmten Server, sondern bindet den *Coda Namensraum* ein. In diesen globaler Namensraum werden alle Freigaben (Volumes) eingehängt. Bindet der Client das Coda Dateisystem ein, sieht er automatisch alle Freigaben in diesem Namensraum. Auf welchem Server sich welches Verzeichnis befindet ist für den Benutzer nicht erkennbar, sondern wird vom Coda Subsystem organisiert.

Weitere Informationen zum Konzept des globalen Namensraumes sind in Abschnitt 2.5.2 zu finden.

2.6.2. Caching

Wie bereits erwähnt, basiert Coda auf der Technik von AFS und erweitert dieses um *high availability* und *disconnected operation*. Neben der Replikation von Servern, die hier nur am Rande erwähnt wird, ist ein Client-seitiger Cache der zentrale Punkt für Hochverfügbarkeit und das Arbeiten ohne Netzwerkverbindung. Der Cache ist ein Bereich auf der Festplatte, der eine festgelegte Größe hat. In diesem kann Coda Dateien und Verzeichnisse vom Server zwischenspeichern. Besteht keine Verbindung zum Server, werden die Daten anstatt vom Server aus dem Cache gelesen und in diesen geschrieben. Nach Wiederherstellung der Netzwerkverbindung überträgt das System die Änderungen zurück auf den Server. Coda unterscheidet dabei nicht, ob der Verlust der Netzwerkverbindung auf einen kurzzeitigen Netzwerkausfall zurückzuführen ist, oder ob der Benutzer den Rechner absichtlich vom Netzwerk getrennt hat.

AFS besitzt ebenfalls einen Cache. Dieser dient jedoch hauptsächlich der Performancesteigerung. Es ist nicht möglich diesen zur Überbrückung von Netzwerkausfällen zu

verwenden (Siehe Abschnitt 2.5.3). Aus diesem Grund ist die Umsetzung unter Coda etwas anders. Neben dem VFS Modul im Kernel existiert ein Cache-Manager. Dieser trägt den Namen *Venus* und sorgt für die Kommunikation mit dem Server, sowie die Aktualität des Cache-Inhaltes. Venus ist ein user-space Prozess, er läuft also ausserhalb des Kernels. Dies führt zu einem sehr kleinen Kernel Modul und erleichtert Portierung und Debugging. [22] (Abb.: 2.3)

Möchte ein Programm Zugriff auf eine Datei im Coda Verzeichnisbaum erhalten, wendet sich dieses, wie unter Linux üblich, an den VFS Treiber des Kernels und damit an das Modul für Coda. Dieses bittet nun Venus um ein Datei-handle. Der Cache-Manager überprüft darauf hin, ob sich die Datei bereits im lokalen Cache befindet. Ist dies noch nicht der Fall (*Cache Miss*), sucht Venus nach einem Server, von dem er die Datei beziehen kann. Nach dem Herunterladen gibt er ein Filehandle zurück, über das ein direkter Zugriff auf die Datei im Cache erfolgen kann. Bei der nächsten Anfrage ist diese Datei bereits vorhanden und muss nicht erneut geladen werden. Wird die Datei nach einem Schreibzugriff geschlossen, überträgt Venus sie unmittelbar an den Server, sofern eine Verbindung besteht. [5]

Zustände des Cache-Managers

Es gibt drei Zustände die der Cache-Manager Venus einnehmen kann (Abb.: 2.4) [36]:

Hoarding Hoarding (Vorrat ansammeln) wird der Zustand genannt, in dem sich Venus bei bestehender Netzwerkverbindung befindet. Venus sorgt in diesem Zustand dafür, dass sich die gewünschten Dateien im Cache befinden und diese aktuell sind.

Emulation Wird die Netzwerkverbindung absichtlich oder unerwartet getrennt, wechselt Venus in den Emulationsmodus. Für den Benutzer ändert sich dabei zunächst nichts.

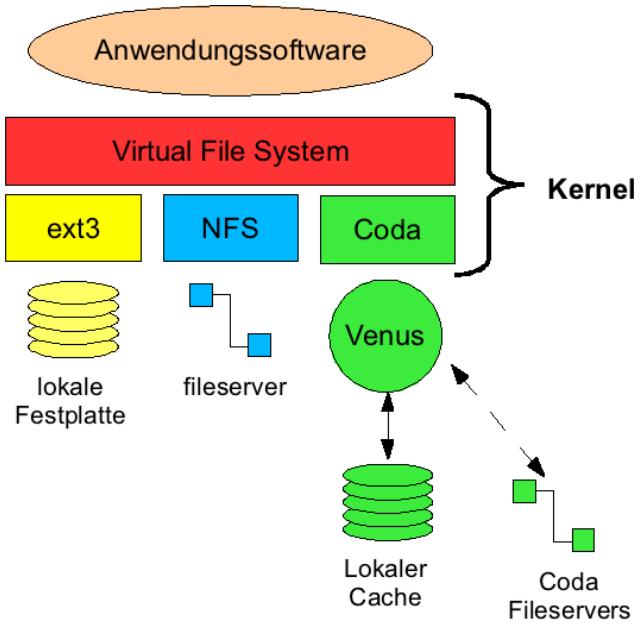

Abbildung 2.3.: Anordnung des Coda-Cache Managers

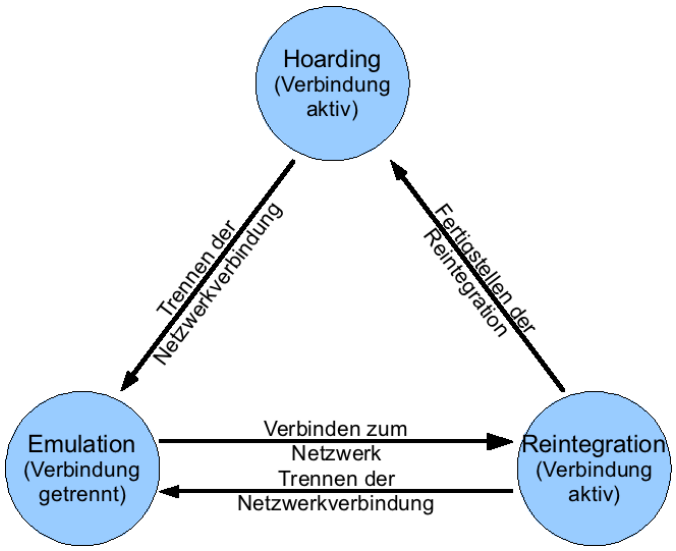

Abbildung 2.4.: Der Coda Cache-Manager

Er kann so weiterarbeiten wie bisher. Im Fall eines Cache-Miss (Datei befindet sich nicht im Cache) wird er mit einem Fehler abgewiesen. Alle Änderungen die in diesem Zustand im Cache durchgeführt werden, schreibt Venus zusätzlich in den Client Modification Log (CML) (Siehe Abschnitt 2.6.4). Dies erleichtert die spätere Reintegration.

Reintegration Nach Wiedererstellung der Netzwerkverbindung wechselt Venus in den Reintegrationszustand. Er versucht die im Emulationsmodus durchgeführten Änderungen, die im CML gespeichert wurden, auf den Server zu übertragen (Siehe 2.6.5). Anschließend geht er in den hoarding Status zurück.

Cache Aktualität

Das erste Problem, das sich bei diesem Konzept stellt ist, dass die im Cache befindlichen Dateien aktualisiert werden müssen, sobald sie auf dem Server geändert wurden. Andernfalls würde der Client eine veraltete Version verwenden.

AFS verwendet eine Methode die *callback-based-caching* genannt wird. Der Server sendet dem Client zusammen mit jeder Datei einen Callback[4]. Ändert sich die Datei auf dem Server, so informiert dieser die Clients durch den sogenannten *callback break*. Beim nächsten Zugriff auf die Datei weiß der Cache-Manager nun, dass er die Datei erneut laden muss. [28]

Bei Coda ist dieses Verfahren aus folgendem Grund nicht einsetzbar: Lädt der Cache-Manager Dateien mit ungültigem Callback erst dann neu herunter, wenn ein Zugriff auf diese geschieht, so sind viele Dateien im Cache veraltet. Dies darf nicht passieren, da bei plötzlichem Verbindungsverlust alle Dateien aktuell sein müssen. Eine Alternative wäre, die Datei sofort nach dem callback break neu herunterzuladen. Es ist allerdings sehr

[4] Auch *Callback-Promise* genannt. Ein Versprechen des Servers, den Client bei Änderung an der Datei zu informieren

wahrscheinlich, dass die selbe Datei mehrmals direkt hintereinander bearbeitet wird. Das Netzwerk würde sehr stark belastet werden, wenn eine Datei bei jeder Änderung auf alle Clients übertragen würde.

Die Entwickler von Coda haben einen Kompromiss gewählt, um diese Probleme zu umgehen. Befindet sich Venus im *hoarding* Zustand, wird in Abständen von 10 Min. ein sogenannter Hoard-Walk durchgeführt. Hierbei lädt Coda unter anderem alle Dateien mit ungültigem Callback herunter. Bei Zugriff auf eine Datei wird diese selbstverständlich sofort aktualisiert. Wird die Netzwerkverbindung zwischen den Hoard-Walks getrennt, kann es trotzdem noch passieren, dass Dateien mit ungültigem Callback im Cache liegen. Diese Dateien werden als „nicht verfügbar“ markiert und liefern einen Fehler beim Zugriff. Verzeichnisse mit ungültigem Callback bleiben jedoch weiterhin gültig.

Auswahl der zu cachenden Dateien

Für den Cache steht eine bestimmte Größe an Speicherplatz auf der Festplatte zur Verfügung. Es ist die Aufgabe des Cache-Managers, die Dateien auszuwählen, die hier vorgehalten werden. Dazu verwendet er einen Algorithmus, der aus den Präferenzen des Benutzers und dessen Nutzungsverhalten eine Priorität berechnet. Bei jedem Zugriff auf eine Datei wird deren Priorität aktualisiert. Dateien, die häufiger verwendet werden, bleiben dadurch längere Zeit im Cache. Ebenso Dateien, auf die erst kürzlich zugegeriffen wurde. Zusätzlich hat der Benutzer die Möglichkeit, über sogenannte *hoard profiles* dem Cache-Manager mitzuteilen, welche Dateien besonders wichtig und welche weniger wichtig sind. Dazu erstellt er eine Textdatei in der er einzelnen Dateien oder Verzeichnisbäumen bestimmte Prioritäten zuordnet.

In den Anfängen von Coda gab es diese Prioritäten nicht. Die Dateien konnten vom Anwender entweder als offline verfügbar, oder nicht offline verfügbar gekennzeichnet werden. Entsprechend holte Venus die Datei in den Cache oder nicht. Eine gängige Fehlan-

nahme in diesem Zusammenhang ist, dass höhere Prioritäten nur die Wahrscheinlichkeit erhöhen, dass eine Datei offline verfügbar ist, während beim alten Konzept sicherstellt werden konnte, dass eine Datei im Cache liegt. Es ist aber nicht gewährleistet, dass der Cache groß genug ist um alle als *offline verfügbar* markierten Dateien zu speichern. Insbesondere wenn die Dateien später stark anwachsen, reicht u.U. der Platz nicht mehr aus. Dann müssen trotzdem Dateien wieder aus dem Cache entfernt werden. Aus diesem Grund ist das Prioritätensystem sinnvoll. [36]

Der Cache-Manager führt, solange er sich im hoarding Zustand befindet, alle 10 Min. einen sogenannten *hoard walk* durch. Dieser sorgt dafür, dass der Cache im *Gleichgewicht* ist. Dies bedeutet, dass es kein Objekt gibt, das eine höhere Priorität hat als die Objekte im Cache. Mit anderen Worten: Die Objekte mit den höchsten Prioritäten befinden sich im Cache. Diese Prioriäten werden aus den impliziten, durch das Nutzungsverhalten vorgegeben und expliziten, durch die in den *hoard profiles* vorgegebenen Prioritäten berechnet. Während des hoard walks besorgt Venus auch die aktuellen Versionen von veralteten Dateien. [36] [22]

2.6.3. Optimistic Replication

Bearbeiten mehrere Benutzer gleichzeitig die selbe Datei, so tritt folgendes Problem auf: Speichert der erste seine Datei ab, so wird diese einfach zum Server übertragen. Beendet der nächste Benutzer seine Arbeit gibt es einen Konflikt.

Viele Dateisysteme, darunter auch AFS, lösen dieses Problem durch einen Mechanismus, der File-Locking genannt wird. Eine Software kann sich exklusiven Zugriff auf eine Datei verschaffen, indem sie den Unix-Befehl *flock* verwendet. Öffnet ein weiterer Benutzer die selbe Datei, kann er diese nur lesen. Nach dem Bearbeiten wird die Sperre wieder aufgehoben.

In einer Umgebung, die kurzzeitige Netzwerkausfälle überbrücken muss, ist dieses Kon-

zept durchführbar. Offlinefähigkeit bedeutet jedoch, dass ein Mitarbeiter eine Menge an Dateien mit nach Hause nimmt um sie zu verändern. Vor dem Verlassen des Büros müsste er also sämtliche Dateien, die er unterwegs bearbeiten wird, sperren. Während seiner Abwesenheit hätte keiner seiner Kollegen die Möglichkeit, die gesperrten Dateien zu bearbeiten. Im produktiven Einsatz ist diese Lösung unpraktikabel. Es erfordet ein hohes Maß an Absprache zwischen den Benutzern und falls sich die Rückkehr eines Mitarbeiters unerwartet verzögert, kann u.U. die Arbeit nicht fortgesetzt werden. Deshalb haben sich die Entwickler von Coda für die sogenannte *Optimistic Replication*[32] entschieden. Dieses Konzept besagt, dass aus oben genannten Gründen auf Locking verzichtet und dadurch das Risiko von Dateikonflikten eingegangen wird [34]. Es ist nun die Aufgabe des Dateisystemes, diese Konflikte zu erkennen und zu behandeln.

2.6.4. Logging

Besteht keine Netzwerkverbindung, wechselt Venus in den Emulations-Zustand. Hierbei erscheint es dem Anwendungsprogramm, als wäre der Server weiterhin erreichbar. Um die Reintegration zu erleichtern, werden Änderungen an Dateien und Verzeichnissen im sogenannten CML aufgezeichnet. Legt der Benutzer beispielsweise eine Datei an, so wird dafür ein Eintrag in den CML geschrieben. Das gleiche gilt, wenn er eine Datei löscht. Mehrere Schreibzugriffe speichert Venus nicht einzeln im Log, sondern legt einen Eintrag *Datei geändert* an. Bei der Reintegration wird die Datei dann komplett auf den Server übertragen und nicht jeder Schreibzugriff einzeln ausgeführt.

2.6.5. Reintegration und Konfliktauflösung

Nachdem die Verbindung zum Netzwerk wiederhergestellt wurde, wechselt Venus in den Reintegrations-Zustand und versucht die lokalen Änderungen auf den Server zu übertragen. Dabei kann es vorkommen, dass eine lokal geänderte Datei in der Zwischen-

zeit von einem anderen Client auf dem Server bearbeitet wurde. Um Inkonsistenzen zu vermeiden, wird ein Transaktionskonzept verwendet. Da Coda die Reintegration für jedes Volume (Abschnitt 2.5.2) einzeln ausführt, können im Fehlerfall alle Änderungen an diesem Volume gleichzeitig rückgängig gemacht werden.

Reintegration

Beim Eintritt in den Reintegrations-Zustand sendet Venus zunächst den CML an den Server. Die Reintegration läuft nun in vier Phasen ab. Im ersten Schritt startet der Server eine Transaktion und sperrt alle im CML referenzierten Dateien für Schreibzugriffe.

In Phase zwei wird neben der Prüfung der Zugriffsrechte eine Konflikterkennung durchgeführt. Diese basiert auf dem Vergleich der sogenannten *storeid*. Dies ist eine Art Versionsnummer, die jedes Objekt besitzt und über die das letzte Update eindeutig festgestellt werden kann. Bei Schreibzugriff auf dem Server verändert dieser die *storeid*. Stimmt diese Zahl im CML mit der auf dem Server überein, so wurde die Datei zwischenzeitlich nicht auf dem Server verändert. Sie darf also mit der neuen Version überschrieben werden.

Bei unterschiedlichen Versionsnummern unterschiedet der Server, ob es sich um eine Datei oder ein Verzeichnis handelt. Tritt ein Datei-Konflikt auf, bricht der Server die Transaktion ab und meldet dem Client den entsprechenden Fehler. Dieser versucht den Konflikt zu beheben. Bei Verzeichnis-Konflikten löst der Server das Problem selbst.

Wurde Phase zwei erfolgreich ausgeführt, erfolgt in Schritt drei das sogenannte *back-fetching*. Der Server holt sich alle notwendigen Dateien vom Client. Die letzte Phase ist die Umkehrung von Phase eins. Hier wird die Transaktion abgeschlossen und alle Sperren aufgehoben.

Konfliktauflösung in Verzeichnissen

Verzeichnisse sind Dateien, die nach einem bestimmten Format aufgebaut sind. Sie enthalten eine Liste mit Dateien und Unterverzeichnissen. Wurde ein Verzeichnis auf beiden Seiten geändert, so besteht zunächst ein Konflikt. Der Server versucht Verzeichnis-Konflikte selbständig aufzulösen. Anhand des CML sieht er, welche Dateien und Verzeichnise angelegt, gelöscht oder umbenannt wurden. Dadurch ist es einfach, die entsprechenden Verzeichniseinträge anzupassen.

Existiert beispielsweise in einem lokalen Verzeichnis ein Eintrag für eine Datei, der auf dem entsprechenden Verzeichnis auf dem Server nicht vorhanden ist, so würde sich die Frage stellen, ob die Datei lokal neu angelegt oder auf dem Server gelöscht wurde. Bei Coda stellt sich diese Frage jedoch nicht, da im CML mitgeschrieben wurde, was passiert ist. Dadurch können die meisten Verzeichnis-Konflikte ohne Interaktion des Benutzers eindeutig aufgelöst werden.

In folgenden Fällen ist der Konflikt nicht selbständig auflösbar:

- Auf beiden Seiten wurde ein Objekt mit dem gleichen Namen angelegt.
- Ein Objekt wurde auf einer Seite verändert, während es auf der anderen Seite gelöscht wurde.
- Attribute eines Verzeichnisses wurden auf beiden Seiten verändert.

Diese Szenarien führen dazu, dass der Server die Reintegration abbricht und dem Client den Fehler meldet.

Konfliktauflösung in Dateien

Konnte die Konfliktfreiheit nicht bewiesen werden, so wird davon ausgegangen, dass sich die Dateien tatsächlich auf beiden Seiten verändert haben. Ein Prinzip von Coda besagt, dass Änderungen niemals verlorengehen dürfen, ohne dass der Benutzer dies

explizit bestätigt. Der Client muss also beide updates miteinander verbinden oder den Benutzer bitten, dies selbst zu erledigen.

Es gibt viele unterschiedliche Dateitypen und -formate. Coda müsste für jedes erdenkliche Format einen eigenen Algorithmus bereitstellen, der zwei Versionen einer Datei zusammenfügen kann. Da dies aufgrund der Vielfalt und ständig wachsenden Zahl von Dateiformaten nicht möglich ist, haben die Entwickler von Coda ein modulares Konzept entwickelt. Coda stellt ein Framework für so genannte Application-specific resolver (ASR) zur Verfügung. Damit können eigene Algorithmen zur Auflösung von Konflikten eingehängt werden. Erkennt der Cache-Manager Venus einen Konflikt, sucht er einen zum Dateityp passenden ASR und startet diesen. Ist keiner vorhanden, oder schlägt er fehlt, markiert Venus die Datei als nicht-aufgelöst. Als Konsequenz daraus, wechselt er nicht zurück in den Hoarding Zustand bis alle Konflikte vom Benutzer per Hand aufgelöst wurden.

Zur Auflösung *per Hand* stellt Coda ein Werkzeug zur Verfügung mit dem der CML schrittweise durchgegangen und die einzelnen Operationen angenommen oder verworfen werden können.

Wahrscheinlichkeit von Konflikten

Die Entwickler von Coda haben Tests an der AFS Infrastruktur der CMU durchgeführt. Sie sind zu dem Ergebnis gekommen, dass in weniger als 0,75% aller Fälle zwei verschiedene Benutzer im Abstand von weniger als einem Tag auf die gleiche Datei zugreifen. [22]

2.6.6. Bewertung

2.6.7. Aktualität von Coda

Die Entwicklung von Coda wurde bereits in den 80er Jahren begonnen und ist mit Hinblick auf die damaligen Anforderungen an der CMU durchgeführt worden. Da sich Linux und Unix Betriebssysteme, sowie allgemein gültige Konzepte wie Netzwerkprotokolle verhältnismäßig langsam ändern, ist Coda immer noch auf dem Stand der Technik. Allerdings haben sich die Anforderungen an ein verteiltes Dateisystem geändert. So ist Coda beispielsweise dafür ausgelegt, dass man das komplette System auf einem Coda Volume ablegt. Hier bringt der Caching Mechanismus natürlich einen sehr großen Performance-Gewinn. Mittlerweile gibt es aber andere Methoden, die zentrale Administration zu vereinfachen und dieses Konzept wird nur noch selten und in reinen Unix-Umgebungen eingesetzt.

Netzwerkausfälle und Offline Arbeiten

Coda ist ein sehr mächtiges Dateisystem, in einigen Bereichen allerdings zu mächtig. Die Entwickler stellen hohe Verfügbarkeit in den Vordergrund. Dabei mischen sie die Überbrückung von temporären Fehlern mit der Offlinefähigkeit. Dies bedeutet, dass selbst Mitarbeiter, die kein Notebook besitzen bzw. niemals ohne Netzwerkverbindung arbeiten, zwangsläufig trotzdem mit *disconnected operation* in Berührung kommen. Das kann zu Verwirrung führen, da der Netzausfall nicht bemerkt wird und auf einmal nur noch die Dateien verfügbar sind, die im lokalen Cache liegen. Da kein Locking-Mechanismus existiert, könnte es im schlimmsten Fall passieren, dass zwei Leute die selbe Datei gleichzeitig editieren. Dadurch tritt ein Konflikt auf, der zunächst behoben werden muss.

Eine mögliche Lösung für dieses Problem wäre, zwischen geplantem und unvorhergese-

henem Netzwerkverlust zu unterscheiden. Man könnte einen Mechanismus einführen, der Dateien nur sperrt, wenn diese zum Schreiben geöffnet werden. Benutzer, die lediglich von kurzzeitigen Netzwerkausfällen betroffen sind, bekommen dadurch keine Probleme mit Inkonsistenzen. Anwender, welche die Offlinefähigkeit nutzen, müssen allerdings darauf achten, dass beim Trennen der Netzwerkverbindung keine Datei gesperrt ist.

Tritt ein Netzwerkfehler auf, würden alle Schreibzugriffe zwar lokal zwischengespeichert, bis die Verbindung wiederhergestellt ist, andere Benutzer könnten die Datei allerdings zwischenzeitlich nicht verändern. Dadurch werden Konflikte ausgeschlossen und es gehen keine Daten verloren.

Caching von großen Dateien

Bei jedem Zugriff wird die Datei komplett in den lokalen Cache geladen. Arbeitet man mit Dateien, die eine Größe von mehreren 100MB haben, hat dies einen deutlichen Performance-Verlust zur Folge. Ausserdem muss der lokale Cache auch genug Platz für diese Dateien bieten.

Konfliktauflösung

Dafür, dass die Wahrscheinlichkeit von Konflikten sehr gering ist, ist der Aufwand, den die Entwickler in deren Auflösung gesteckt haben, sehr hoch. Das modulare Konzept, um Dateien zusammenzufügen, wird wohl kaum jemand wirklich nutzen. Es ist fast unmöglich, Änderungen in zwei Dateien, die in einem Binärformat vorliegen (z.B. Word Dokumente, Excel Tabellen, Bilder) automatisch zusammenzufügen. Sehr gut dagegen funktioniert dies bei Plain-Text Dateien mit Programmiersprachen-Code. Jedoch greifen Software-Entwickler in der Regel auf andere Systeme, wie CVS oder Subversion, zurück, welche ebenfalls Mechanismen zum Zusammenfügen von Dateien, sowie eine

eingebaute Versionsverwaltung besitzen.

Es würde ausreichen, dem Benutzer bei einem Konflikt diese beiden Dateien zur Verfügung zu stellen und ihn bitten, diese per Hand zusammenzufügen. Dies schützt auch vor Datenverlusten, die bei Fehlern im Zusammenfüge-Algorithmus auftreten könnten.

Daten persistenz

Ein Artikel in Wikipedia [44] informiert über einen weiteren Schwachpunkt von Coda. Dateien werden erst beim Schließen auf den Server kopiert (persistent gemacht). Wird eine Datei permanent offen gehalten, so werden die Änderungen nie öffentlich verfügbar. In [44] heißt es:

> Dies widerspricht den üblichen UNIX-artigen semantiken, dass geschriebene Daten früher oder später ohne weiteres Zutun der Applikation den persistenten Speicher (die Festplatte) erreichen.

Ausserdem heißt es:

> Das bedeutet, dass Änderungen auf einer Datenbank oder Logfile-Einträge nicht erhalten bleiben, sollte der Coda-Client einmal nicht ordnungsgemäß heruntergefahren werden (z. B. Strom-Ausfall bzw. leerer Akku).

Dies ist so allerdings nicht richtig. Die Daten gehen nicht verloren, da sie sich ja trotzdem im lokalen Cache befinden. Nach dem Neustart des Systems werden diese auf den Server reintegriert.

2.6.8. Verbreitung und Einsetzbarkeit

Coda existiert bereits seit Ende der 90er Jahren in weitgehend stabilen Versionen. Trotzdem ist es nicht sehr weit verbreitet.

Ein Grund dafür ist, dass die meisten Firmen Windows-Betriebssysteme einsetzen. In Homogenen Windows-Umgebungen, so wie Heterogenen Umgebungen in denen Windows Maschinen vorhanden sind, wird meistens SMB/CIFS (Abschnitt 2.4) eingesetzt. Coda existierte ursprünglich nur für Linux. Die Windows-Implementierungen von Client und Server sind noch relativ neu und ungetestet. Die Installation ist nicht ganz problemlos und auch im Betrieb erweist sich diese Portierung als nicht besonders stabil.

Ein weiteres Problem ist, dass die Benutzerfreundlichkeit zu wünschen übrig lässt. Im Internet sind nirgends Ansätze für eine grafische Oberfläche zu finden, die den Anwender auf der Client Maschine unterstützen. Wünschenwert wäre hier eine Anzeige, die über den Zustand des Cache-Managers informiert (Abschnitt 2.6.2). Ausserdem müsste der Benutzer bei Konflikten unmittelbar informiert werden und eine eine einfache Möglichkeit haben, diesen aufzulösen. Um hoard profile zu definieren, muss man eine Textdatei editieren. Dazu ist es notwendig, deren Struktur zu kennen und einhalten. Dies ist für einen „normalen“ Benutzer ohne spezielle Ausbildung nicht zumutbar. Auch hier fehlt eine komfortable Oberfläche.

Etwas stutzig macht ein Eintrag in der Coda FAQ [39]. Unter der Frage „How stable and usable is Coda?“ heißt es:

> According to the previous entry from 1998, Coda wasn't ready for production use. What is the current status?
>
> I'd say a small userbase (20-30 users) and a few servers are pretty workable. Such a setup has been running here at CMU for the past couple of years without significant disasters.
>
> Don't expect to easily handle terabytes of data or a large group of non-technical oriented users.

Dieser Aussage zufolge ist Coda momentan noch nicht produktiv einsetzbar. Leider gibt es keine Angabe über das Jahr dieser Information. Die Entwicklung scheint noch

in Gang zu sein. Es gibt jedes Jahr mehrere Releases von Coda und es sind auch einige aktuelle Einträge im Ticketsystem vorhanden.

Ob sich die Verbreitung von Coda in der nächsten Zeit ändert bleibt abzuwarten. Auf der Website [4] ist jedenfalls vermerkt:

> The current activities with Coda are mostly aimed at making this very good file system widely available, and a network file system of choice.

Es gibt allerdings auch hier keine Angabe darüber, wie lange dies schon dort steht.

Sollte Coda tatsächlich einmal das halten, was es verspricht, ist es auf jeden Fall eine Alternative zu AFS und CIFS.

2.7. InterMezzo

Das InterMezzo Projekt wurde von Peter J. Braam an der CMU ins Leben gerufen. Er war zuvor beim Coda-Projekt (Siehe Abschnitt 2.6) leitend tätig und u.A. mit der Portierung auf Windows betraut. InterMezzo ist OpenSource und unter der GPL[16] verfügbar.

Ziel von InterMezzo war es, ein hoch verfügbares verteiltes Dateisystem mit Offline-fähigkeit zu schaffen. Es sollte ähnliche Funktionalität wie Coda bieten, allerdings wesentlicher einfacher und mit weniger Programmcode realisiert werden. Ausserdem sollte möglichst auf offenen Standards aufgesetzt werden [7]. InterMezzo wurde stark durch Coda angeregt. Dies sieht man auch am Namen, denn *Intermezzo* (ital. Zwischenspiel) ist ebenso wie *Coda* ein gebräuchlicher Ausdruck in der Musik.

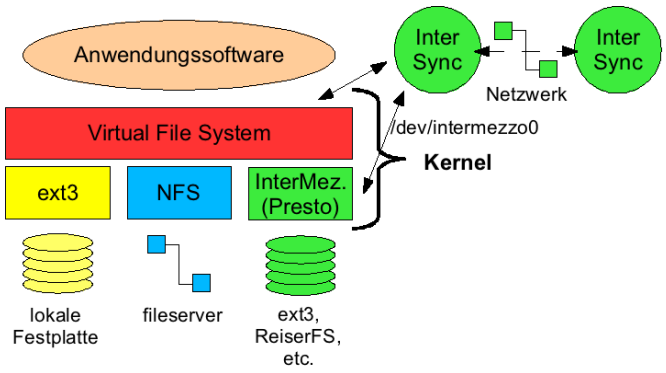

Abbildung 2.5.: InterMezzo und Intersync im System

2.7.1. Globaler Namensraum

Wie bei Coda und AFS gibt es auch bei Intermezzo einen globalen Namensraum in den sämtliche Verzeichnisse eingehängt sind. Während der Client bei anderen Dateisystemen immer die Wurzel einbinden muss, kann der Intermezzo Client auch Teilbäume einhängen, deren Wurzel die Wurzel eines Volumes ist. Weitere Informationen zum Konzept des globalen Namensraumes gibt es in Abschnitt 2.5.2.

2.7.2. Aufbau von InterMezzo

InterMezzo besteht aus zwei Komponenten. Dem InterMezzo Kernel-Modul (genannt Presto) und InterSync, einem Prozess, der dafür sorgt, die Daten aktuell zu halten. Datenbasis von InterMezzo ist ein Journaling Dateisystem (z.B. ext3, ReiserFS). Das InterMezzo Kernel Modul bildet eine Zwischenschicht, zwischen dem VFS und dem Dateisystem, sodass jeder Zugriff „durch“ InterMezzo geht. InterSync läuft ausserhalb des Kernels, kann aber über ein spezielles Device (/dev/intermezzoX) mit diesem kommunizieren. (Abb.: 2.5)

Kernel Modul

Das Kernel Modul ist, wie bei Coda, sehr einfach gehalten. Da es zwischen dem VFS und dem Dateisystem sitzt, kann es sämtliche Änderungen aufzeichnen. Seine Aufgabe ist es, diese in den Kernel Modification Log (KML)[5] zu schreiben, um sie schnell auf die anderen Systeme zu übertragen. Ausserdem sorgt er dafür, dass ein Zugriff so lange blockiert wird, bis die Daten zur Verfügung stehen und verteilt Sequenznummern an die Dateien. Informationen über Dateiversionen und der KML werden in versteckten Verzeichnissen in der darunterliegenden Datenbasis..

InterSync

InterSync ist die Synchronisationskomponente von InterMezzo. Sowohl auf dem Server als auch auf dem Client läuft eine Instanz von InterSync. Auf der Client Seite ähnelt er in seiner Funktionsweise dem Cache-Manager Venus bei Coda. InterSync kommuniziert über das *HTTP* Protokoll. Dies hat den Vorteil, dass vorhandene Techniken wie SSL mit wenig Aufwand genutzt werden können.

Im passiven Modus läuft nur auf dem Fileserver ein InterSync-HTTP-Server[6]. In bestimmten Zeitabständen holt sich der Client InterSync Prozess den KML, in dem alle Änderungen im Dateisystem aufgelistet sind, vom Server. Er arbeitet diesen durch und führt die Änderungen im lokalen Dateisystem (dem Cache) durch. Geänderte Dateien holt er sich ggf. vom Server. Verändert der Benutzer eine Datei im lokalen Cache, sendet InterSync den KML, sowie die geänderten Dateien an den Server.

[5] Der KML enthält eine Liste aller Änderungen, die am Dateisystem vorgenommen wurden. Es werden dabei nur die geänderten Attribute mitgeschrieben (z.B. timestamp). Der Inhalt der Dateien wird nicht aufgezeichnet, da jede Datei bei der Reintegration immer komplett übertragen wird.

[6] InterSync arbeitet mit allen gängigen HTTP Servern zusammen, da es als Common Gateway Interface (CGI) Programm läuft

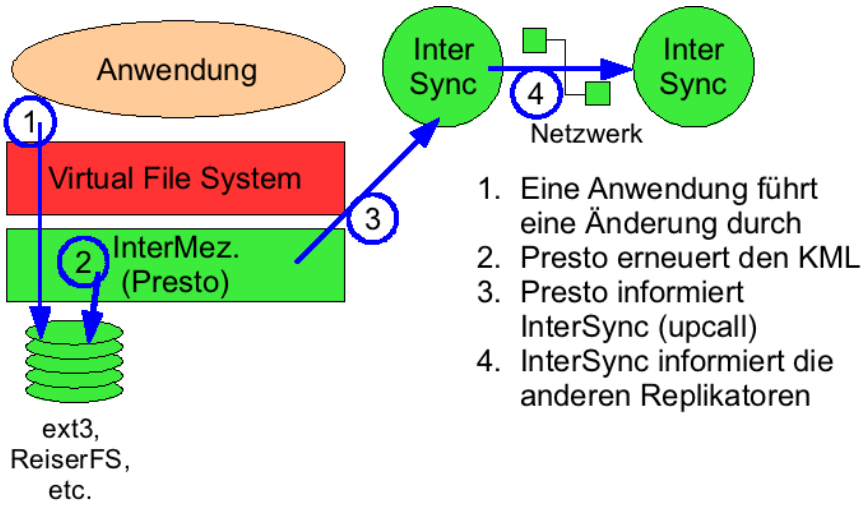

Abbildung 2.6.: Informieren der Client/des Servers, bei Änderung des KML

InterSync unterstützt ausserdem einen aktiven Modus. Bei diesem läuft auch auf dem Client ein InterSync-HTTP-Server. Sobald sich eine Datei und damit der KML auf dem Server ändert, informiert Presto durch sogenannte *upcalls* den InterSync Prozess. Dieser teilt dem Client mit, dass ein neuer KML vorliegt. Der Client kann nun, wie im passiven Modus, den KML und die Dateien abholen. Ändert sich eine Datei auf dem Client, kehren sich die Rollen von Client und Server um (Abb.: 2.6). InterSync kann auch dazu genutzt werden, Server zu replizieren.

2.7.3. Erkennen und Auflösen von Konflikten

Die Konflikterkennung läuft nach dem gleichen Schema ab wie bei Coda. Hat sich die Versionsnummer einer Datei auf dem Server seit dem letzten Update nicht verändert, so liegt kein Konflikt vor (Siehe Abschnitt 2.6.5). Der Algorithmus, den InterSync verwendet, um Konflikte aufzulösen, wird in [6] beschrieben und bewiesen. InterSync bietet keine Möglichkeit Dateien zusammenzufügen, wie dies bei Coda der Fall ist. Kann InterSync einen Konflikt nicht anhand des KML auflösen, hat eines der beiden Repli-

kate Vorrang. Die Konflikt-Datei wird vom bevorzugten auf das benachteiligte Replikat kopiert. Damit keine Daten verloren gehen, wird die zweite Datei nicht überschrieben, sondern vorher umbenannt. Der Benutzer muss diesen Konflikt nun per Hand beheben.

Dies hat den Vorteil, dass die Datei sofort auf allen Systemen synchron ist, aber trotzdem keine Daten verloren gehen. Es gibt drei Regeln, aus denen InterSync auswählt welche der beiden Replikate Vorrang hat. Jede Regel ist für einen bestimmten Einsatzzweck bestimmt.

Mobile policy Synchronisation zwischen einem mobilen Gerät (z.B. Notebook) und einem Server. Die Version vom Server hat Vorrang und wird auf den Client (mobiles Gerät) kopiert. Die Version auf dem Client wird verschoben.

HA(high availability) policy Synchronisation zwischen Servern, welche die Daten redundant halten. Bei Zurückerhalt der Verbindung wird ein Server als *active node* der andere als *failed node* eingestuft. In diesem Fall hat die *active node* Vorrang.

Re-synchronization policy Damit der KML nicht zu groß wird, wirft der Server ältere Einträge von Zeit zu Zeit weg und erstellt einen Synchronisation Modification Log (SML). Dies ist ein vereinfachter KML. Er besteht aus einer Liste der Dateien und Verzeichnissen, die zum Zeitpunkt seiner Erstellung existieren. Die Re-synchronization policy wird immer dann angewendet, wenn zum ersten Mal synchronisiert wird, oder die letzte Synchronisation länger zurückliegt als der älteste Eintrag aus dem aktuellen KML. In diesen Fällen fürt InterSync vor dem KML den SML aus um das Dateisystem auf den Stand vor dem ersten KML Eintrag zu bringen.

2.7.4. Ausblick

Wie anhand der InterrMezzo Website [8] zu erkennen ist, wurde die Weiterentwicklung eingestellt. Der letzte Eintrag im Bereich *News* stammt aus dem Jahr 2002 und das neueste Release, das von SourceForge.net bezogen werden kann ist die Version 1.0-beta1, welche aus dem Jahr 2000 stammt.

Peter J. Braam, der frühere Leiter des InterMezzo Projektes hat eine Firma mit dem Namen Cluster Filesystems Inc. gegründet und arbeitet mit seinem Team nun an dem Dateisystem *lustre* (Abschnitt: 2.8).

In [7] schreibt Peter J. Braam, dass es wohl relativ einfach wäre, InterMezzo zumindest auf Windows NT und 2000 zu portieren, da er dies bereits mit dem Coda Filesystem gemacht hatte. Allerdings wurde dies nicht umgesetzt.

2.7.5. Bewertung

InterMezzo dürfte vielen Ansprüchen genügen. Es ist nicht mit unnötigen Features überladen, sondern konzentriert sich auf das wesentliche. Positiv fällt auch auf, dass in den meisten Fällen das Rad nicht neu erfunden, sondern auf existierende Standards zurückgegriffen wurde. Leider gibt es keine Implementierung für andere Betriebssysteme ausser Linux. Das größte Problem ist die fehlende Möglichkeit zur Auswahl der Dateien, die offline verfügbar sein sollen. InterSync kopiert prinzipiell alle Dateien auf den Client. Bei großen Datenmengen ist dies unbrauchbar, da die Kapazität des mobilen Gerätes in der Regel nicht ausreichend ist. InterMezzo scheint eher auf Server-Replikation ausgelegt zu sein.

InterMezzo und speziell InterSync haben keinen stabilen Zustand erreicht. So lange das Projekt nicht weitergeführt wird, ist es deshalb für den produktiven Einsatz nicht zu empfehlen.

2.8. Lustre

Lustre ist ein junges und leistungsfähiges verteiltes Dateisystem für sehr hohe Anforderungen. Es wird hauptsächlich von *Cluster File Systems, Inc (CFS)* entwickelt. Diese Firma wurde 2001 von Peter J. Braam gegründet, der zuvor die Entwicklung des InterMezzo Filesystem geleitet hatte. CFS bezeichnet sich selbst als weltweit führend in Design und Entwicklung von high-performance cluster file system technology [9].

Das Datenblatt [11], das von der Website [9] bezogen werden kann, informiert über die Vorzüge von Lustre. Hierzu zählen Ausfallsicherheit, Performance und hohe Skalierbarkeit. Es wird in Größenordnungen von zehntausenden Clients und petabytes[7] an Daten gesprochen.

Lustre ist OpenSource und unter der GPL[16] verfügbar. CFS bietet allerdings kostenpflichtigen Support an. Ausserdem behält sich die Firma laut FAQ auf ihrer Website [9] vor, verschiedene Portierungen nicht offen verfügbar zu machen. Auf der Roadmap [12] ist bei den geplanten Implemeniterungen für Windows und Mac OS X vermerkt, dass diese proprietär und nicht frei verfügbar sein werden.

2.8.1. Aufbau und Funktionsweise

Das Lustre Filesystem hat den Aufbau eines Clusters. Sogenannte Metadata Server übernehmen das Cluster-Management. Diese halten Informationen über sämtliche Dateien bereit und wissen, auf welchen Rechnern welche Daten zu finden sind. Bei jedem Zugriff auf eine Datei kontaktiert der Client zuerst einen Metadata Server. Dieser weist ihm einen Rechner zu, auf dem diese Datei gespeichert ist. Der weitere Datenaustausch erfolgt dann direkt zwischen Client und Dateiserver.

[7] 1Petabyte = 10^15Byte

2.8.2. Disconnected Operation

Obwohl die Entwickler von Lustre früher an den Dateisystemen InterMezzo und Coda gearbeitet haben, welche beide Wert auf Disconnected Operation legen, unterstützt Lustre dieses Feature nicht.

Peter J. Braam begründet dies in einer Antwort auf eine Frage an die *Linux Filesystem Development Mailinglist* wie folgt (Siehe: [3]):

> The Lustre architecture does allow for modular extensions that enable disconnected operation, but no customers have asked for it yet. [...] Disconnected operation would involve a client cache, which will be many many times slower than the distributed network infrastructure when the file sizes exceed what can be cached in memory on the client. This is unusual but quite important for supercomputing and some industrial applications.

Unter [23] heißt es, dass Disconnected Operation in Lustre ab Version 2.0 geplant sei. Ein Blick auf die Roadmap [12] zeigt, dass es mittlerweile auf Version 3.0 verschoben wurde.

2.9. Zusammenfassung

Alle in Kapitel 2 beschriebenen Dateisysteme haben Vor- und Nachteile. In Abschnitt 1.3 wurden Kriterien zum Vergleich von verteilten Dateisystemen aufgeführt. Die Dateisysteme sollen nun anhand dieser bewertet werden.

2.9.1. Netzwerk- und Offlinetransparenz

Alle Dateisysteme bieten Transparenz für den Benutzer. Aus Anwendersicht gibt es keinen Unterschied, ob er auf einem lokalen oder einem entfernten Dateisystem arbeitet.

Ausserdem muss er sich nicht umstellen, wenn er zwischen Online- und Offlinemodus wechselt. Ausnahmen zu diesem Verhalten bilden Unison und SyncToy. Diese sind, wie bereits erwähnt, keine Dateisysteme, sondern ermöglichen es lediglich, Verzeichnisse abzugleichen.

2.9.2. Automatische Synchronisation

Während bei Unison und SyncToy der Abgleich per Hand gestartet werden muss, machen die Offlinefilesysteme dies in gewissem Maße automatisch. Nahezu optimal im Bezug auf Aktualität ist dies bei InterMezzo gelöst. Sobald sich eine Datei auf dem Server verändert, werden alle Clients darüber benachrichtigt. Diese beginnen darafhin sofort mit der Replikation. Zwar ist dadurch gewährleistet, dass stets alle Daten aktuell sind, allerdings wird das Netzwerk sehr stark belastet.

Microsoft ist mit seiner Offlinefilesystem Lösung (Abschnitt 2.4) einen Mittelweg gegangen. Beim An- und Abmelden wird eine vollständige, im Leerlauf jedoch nur eine teilweise Synchronisation durchgeführt. Grund dafür ist, dass bei einer vollständigen Synchronisation der komplette Verzeichnisbaum durchsucht werden muss, was sehr zeitaufwändig ist.

Es ist dadurch gewährleistet, dass alle Dateien, die der Benutzer im verbundenen Zustand verwendet, im Cache nicht älter sind als sein letzter Zugriff. Allerdings können andere Dateien durchaus älter sein. Je nach Anwendungsfall ist dies mehr oder weniger problematisch. Was hier fehlt ist die Möglichkeit einen vollständigen Abgleich per Hand zu starten.

Eine sehr gute Lösung wurde bei Coda (Abschnitt 2.6) gewählt. Bei Änderung einer Datei meldet der Server durch den sogenannten *Callback Break* dem Client, dass eine Datei nicht mehr aktuell ist. Erst beim Hoard-Walk, der alle 10 Min. durchgeführt wird, holt sich der Client die Datei vom Server. Bei einem Dateizugriff im Verbundenen Zu-

stand wird die Datei sofort aktualisiert. Es ist also keine Datei älter als 10 Min. und gleichzeitig wird CPU- und Netzwerklast reduziert. In manchen Fällen kann es möglich sein, dass 10 Min. alte Dateien nicht aktuell genug sind. In diesem Fall kann man jederzeit einen manuellen Hoard-Walk ausführen.

2.9.3. In-Time Synchronisation

Trennt der Benutzer die Verbindung planmäßig, setzt er voraus, dass alle Dateien im Cache aktuell sind. Microsoft Offlinefiles führt beim Abmelden vom System einen *full sync* durch. Dadurch ist sichergestellt, dass alle lokalen Dateien aktuell sind. Beim Wechsel in den Ruhezustand (Hibernate) wird dies nicht durchgeführt und *full sync* manuell zu starten ist nicht möglich. Man muss den Rechner also immer komplett herunterfahren oder sich zumindest abmelden.

Die Dateisysteme Coda und InterMezzo verwenden eine elegantere Lösung. Die Cache-Manager sorgen dafür, dass die Dateien zu jedem Zeitpunkt aktuell, bzw höchstens 10 Min. alt sind. Der Benutzer spart dadurch die Synchronisationszeit und kann die Verbindung zu jeder Zeit trennen.

2.9.4. Freiheit von Inkonsistenzen

Alle Offlinedateisysteme bieten die Möglichkeit, dass der Benutzer bei unvorhergesehenem Netzwerkverlust seine Arbeit an einer Datei beenden und speichern kann. Ist der Server wieder verfügbar, werden die Dateien reintegriert. Bei Dateisystemen wie beispielsweise NFS gibt es diese Möglichkeit nicht. Auch SMB unterstützt dies nicht, wird allerdings durch den Einsatz der Windows Offlinefiles um diese Funktion erweitert.

2.9.5. Replikation

Server-Replikation ist bei allen vorgestellen Offlinefilesystemen möglich. Bei der Microsoft Lösung ist die Replikation Sache des DFS.

2.9.6. Skalierbarkeit

Coda, InterMezzo und DFS verwenden einen globalen Namensraum (Abschnitt 2.5.2). Dadurch ist es möglich, Volumes ohne Ausfälle hinzuzufügen und zu entfernen. Bei AFS ist es sogar möglich, während dem Betrieb ein aktives Volume auf einen anderen Server zu verschieben [15].

2.9.7. Benutzerfreundlichkeit

Im Punkt der Benutzerfreundlichkeit haben alle Dateisystemen sehr große Schwächen. Für Coda und InterMezzo gibt es keine grafische Benutzeroberfläche. Von Reintegration und Konflikten bekommt der Benutzer nichts mit und muss zum Auflösen Programme auf der Kommandozeile verwenden. Die Auswahl der zu synchronisierenden Dateien geschieht über Konfigurationsdateien, was einem normalen Benutzer in der Regel nicht zumutbar ist.

Die beste Oberfläche bietet Windows Offlinefiles, wobei auch diese noch einige Schwachstellen hat. Um einen Ordner offline verfügbar zu machen, muss der Benutzer lediglich über das Kontextmenü des Ordners einen Haken setzen. Treten bei der Reintegration Konflikte auf, wird dies dem Benutzer sofort gemeldet. Dieser kann dann sehr komfortabel auswählen, welche Dateien er behalten möchte und welche überschrieben werden sollen. Störend ist das Synchronisationsfenster, das bei jedem Abgleich automatisch angezeigt wird. Bei Fehlern, die auf eine nicht vorhandene Verbindung zum Server zurückzuführen sind, bleibt dieses Fenster stehen und muss per Hand geschlossen werden.

Dies passiert auch dann, wenn ein Teil der Server verfügbar ist.

2.9.8. Anpassung an die vorhandene Infrastruktur

Da die Tools Unison und SyncToy als reine Client-Software verwendet werden können, besteht zunächst kein Problem, diese in einem bestehenden Netzwerk einzuusetzen. Um die volle Funktionalität von Unison nutzen zu können muss man jedoch auch auf dem Datei-Server die Unison Software installieren. Es müsste dann aber ein „normales“ Dateisystem zusätzlich eingesetzt werden. Für die Anwender ist es verwirrend mit zwei Lösungen parallel zu arbeiten, deshalb wird hiervon für diesen Einsatzzweck abgeraten.

Das Dateisystem AFS wurde ursprünglich für Unix Systeme entwickelt. Dadurch lässt es sich auch relativ einfach in eine Unix Umgebung integrieren. Die Rechteverwaltung übernimmt AFS selbst, Authentifizierung kann über Kerberos geschehen und ist damit so flexibel wie der Kerberos Server. Auch die Integration in eine Windows Umgebung ist laut [15] relativ einfach. Zwar arbeitet der AFS Client nicht mit dem Kerberos Client von Microsoft zusammen, sondern benötigt die MIT Kerberos Version, die beiden können jedoch Informationen austauschen, sodass trotzdem single Sign-On möglich ist. Alle Server müssen jedoch auf Unix Systemen laufen. Da in größeren Netzwerken ohnehin dedizierte Dateiserver eingesetzt werden, ist dies allerdings ein kleineres Problem.

Coda sollte ursprünglich eine erweiterte Version von AFS sein, also auch sämtliche Features übernehmen. Wie ausgereift die Kerberos Implementierung ist, lässt sich allerdings nicht bewerten ohne es zu testen. Ausserdem arbeitet der Windows-Client noch nicht zufriedenstellend. Bei einer homogenen Linux Umgebung sollte es wenige Probleme geben.

Auch InterMezzo unterstützt Authentifizierung über Kerberos. Allerdings gibt es auch hier noch einige Sicherheitslöcher und auch die Stabilität ist fragwürdig, da InterMezzo

nichtmehr weiterentwickelt wird. Ausserdem kann es nur unter Linux betrieben werden.

Die Microsoft DFS Implementierung funktioniert nur in einer homogenen Windows Umgebung optimal. Allerdings lassen sich die Dateiserver und mittlerweile auch der DFS-Root Server durch Samba auf Unix Rechnern betreiben. Samba unterstützt Kerberos und es ist auch relativ einfach möglich, einer Windows 200x Domäne beizutreten oder Benutzer Authentifizierung über LDAP zu konfiguieren.

Der dritte Bestandteil, die Offlinefiles, gibt es nur für Windows. Unter Linux ist Offlinefähigkeit mit Samba nicht möglich. Die Erfahrung zeigt, dass das SMB Subsystem sehr instabil reagiert, wenn man die Netzwerkverbindung auch nur kurz trennt, während eine Freigabe eingebunden ist. Ein Ziel unseres Projektes (Siehe Abschnitt 3) ist es, dieses Manko zu beseitigen.

Alle Dateisysteme lassen sich weitgehend in bestehenden Unix oder Windows Umgebungen einsetzen, sofern die Möglichkeit besteht, Kerberos zur Benutzer Authentifizierung zu verwenden. Es ist allerdings nicht zu empfehlen, da die Implementierungen sehr experimentell sind und die Installation und Konfiguration sehr wahrscheinlich mit großem Aufwand verbunden sind. Nur SMB und AFS sind gut dokumentiert und stabil, unterstützen aber leider disconnected operation nicht. Meine Kenntnis von Novell Systemen reicht nicht aus, um zu bewerten wie gut die Dateisysteme mit diesen zusammenarbeiten.

Kapitel 3.

Design eines eigenen Offline Filesystems

Aufgrund des Vergleiches der Vor- und Nachteile der bereits bestehenden Dateisysteme, die disconnected Operation unterstützen, soll nun ein eigenes verteiltes Dateisystem entwickelt werden, das alle Vorteile vereint. Der Arbeitstitel des Projektes lautet Ohm File-System (OFS).

3.1. Anforderungen

Die Anforderungen an unser Dateisystem orientieren sich im Wesentlichen an den Punkten, die in Abschnitt 1.4 aufgeführt sind. Besonders wichtig ist die Benutzerfreundlichkeit. Es muss zumindest eine Schnittstelle erstellt werden, die es später ermöglicht, mit wenig Aufwand Benutzeroberflächen zu schreiben.

Es gibt einige brauchbare Netzwerk-Dateisysteme, die bereits produktiv eingesetzt werden. Dazu zählen SMB/CIFS, NFS und AFS. Es ist nicht sinnvoll, ein weiteres „besseres“ Dateisystem zu erstellen. Zum einen ist dies mit einem sehr hohen Entwicklungs-

aufwand verbunden, zum anderen dauert es sehr lange Zeit, die notwendige Akzeptanz für dieses Dateisystem zu schaffen. Aus diesem Grund soll OFS auf den bestehenden Dateisystemen aufsetzen und diese lediglich um Offlinefähigkeit erweitern. Anders als bei Coda, welches „AFS mit Offlinefähigkeit“ ist, wird OFS keine Neuimplementierung bestehender Software, sondern eine Erweiterung. Es sollen möglichst wenige Modifikationen an den darunterliegenden Dateisystemen vorgenommen werden, sodass im Optimalfall mit jedem beliebigen Dateisystem zusammengearbeitet werden kann.

Ziel des OFS ist es, Benutzern eine einfache Möglichkeit zu geben, Dateien aus dem Netzwerk unterwegs mitzunehmen und diese bei der Rückkehr in die Firma wieder mit dem Server abzugleichen. Vor dem Verlassen des Netzwerkes teilt der Benutzer dem System mit, welche Dateien und Verzeichnisse er unterwegs benötigt. Es ist nicht die Aufgabe dieses Projektes die Verfügbarkeit von Daten zu erhöhen und kurzzeitige Netzwerkausfälle zu überbrücken. Dies ist Sache des eigentlichen Dateisystems.

Zunächst wird das Dateisystem nur für Linux entwickelt. Portierungen auf andere Unix-Systeme gestalten sich relativ einfach.

3.2. Design-Konzepte

3.2.1. Begriffsklärung

OFS soll sich zwischen das VFS von Linux und einem Netzwerk-Dateisystem anordnen. Freigaben werden dann nicht mehr direkt in das Dateisystem eingebunden, sondern nehmen den „Umweg“ über das OFS. In diesem Fall sprechen wir hier von *OFS-Managed-Mountpoint*.

Den Zustand einer Netzwerkfreigabe, zu der eine Verbindung besteht, nennen wir *online*. Ist keine Verbindung vorhanden, befindet sie sich im Zustand *offline*.

3.2.2. Einteilung des Verzeichnisbaumes

AFS und Microsoft DFS verwenden einen globalen Namensraum (Siehe: 2.5.2). Es gibt also aus Sicht des Clients nur einen Mountpoint. Dieser ist intern jedoch in Volumes eingeteilt. Bei den klassischen Dateisystem SMB und NFS werden die Freigaben von verschiedenen Servern einzeln im Dateisystem des Clients eingebunden.

Auch bei unserem Dateisystem ist eine Einteilung des Verzeichnisbaumes in Unterbäume notwendig. Diese Abschnitte werden im Folgenden *Cache-Partitions* genannt. Es ist sinnvoll, den Cache aufzuteilen, da dies eine flexiblere und schnellere Reintegration ermöglicht. Ausserdem kann es passieren, dass bei Verbindung mit einem Netzwerk nicht alle Dateien gleichzeitig verfügbar sind. Dies kann beispielsweise auftreten, wenn man in verschiedenen Netzwerken arbeitet. Cache-Partitions sind unabhängige Verzeichnisbäume. Sie haben eine Wurzel und dürfen sich nicht überschneiden oder verschachtelt sein. Reintegration und Update laufen separat, d.h. es müssen nicht alle Cache-Partitions gleichzeitig im online oder offline Zustand sein. Es ist dagegen nicht möglich, dass nur ein Teil einer Cache-Partition online ist.

Cache-Partitions sollten weder zu groß, noch zu klein gewählt werden. Für die Einteilung betrachten wir die folgenden Möglichkeiten (Abb.: 3.1):

Basierend auf Mountpoints Jeder OFS-Managed-Mountpoint bildet eine Cache-Partition. Da bei SMB jeder Mountpoint genau einem Server zugeordnet ist, wird sichergestellt, dass entweder die gesamte Freigabe verfügbar ist, oder nicht. AFS verwendet allerdings nur einen einzigen Mountpoint. Es gibt also nur eine einzige sehr große Cache-Partition für das gesamte Dateisystem, was unserem Konzept widerspricht. Da das AFS intern in Volumes eingeteilt ist, die dynamisch zur Laufzeit ein- und wieder ausgehängt werden können, ist ausserdem nicht sichergestellt, dass stets die gesamte Cache-Partition online oder offline ist.

Basierend auf der Einteilung des darunterliegenden Dateisystems Je nach

Dateisystem werden die Cache-Partitions unterschiedlich gebildet. Bei Dateisystemen wie SMB werden die Mountpoints verwendet, da dies die kleinste Einteilung ist. Bei Dateisystemen, die Volumes unterstützen, wie AFS, geschieht die Aufteilung nach diesen. Ein Nachteil dieses Konzeptes ist, dass sehr tief in das darunterliegende Dateisystem eingegriffen werden muss. OFS muss die Volume-Aufteilung des darunterliegenden Dateisystems erkennen, die normalerweise auf dieser Ebene nicht mehr sichtbar ist. Da Volumes verschachtelt sind, enstehen in diesem Fall auch verschachtelte Cache-Partitionen. Dies kann jedoch behoben werden, wenn man die Volumes unabhängig vom Verzeichnisbaum und den darin eingehängten Unter-Volumes betrachtet.

Benutzerdefinierte Aufteilung In den Anforderungen (Abschnitt 3.1) wurde festgelegt, dass der Benutzer selbst die Verzeichnisbäume auswählt, die er offline verfügbar haben möchte. Möglich wäre, aus jedem vom Benutzer ausgewählen Verzeichnisbaum eine Cache-Partition zu machen. Die übrigen Dateien müssen ohnehin nicht im Cache abgelegt werden, deshalb können diese ignoriert werden. Es ist dadurch automatisch sichergestellt, dass sich Cache-Partitionen nicht überschneiden. Da sich vom Benutzer gewählte Verzeichnisbäume prinzipiell auch über mehrere Volumes erstrecken können ist allerdings nicht sichergestellt, dass stets die gesamte Cache-Partition verfügbar ist.

Wir werden die letzte Methode in unserem Dateisystem umsetzen. Ein Grund dafür ist die Übersichtlichkeit, da sich die Cache-Partitionen mit den vom Benutzer gewählten Verzeichnisbäumen decken. Ausserdem ist die Implementierung einfacher und strukturierter möglich.

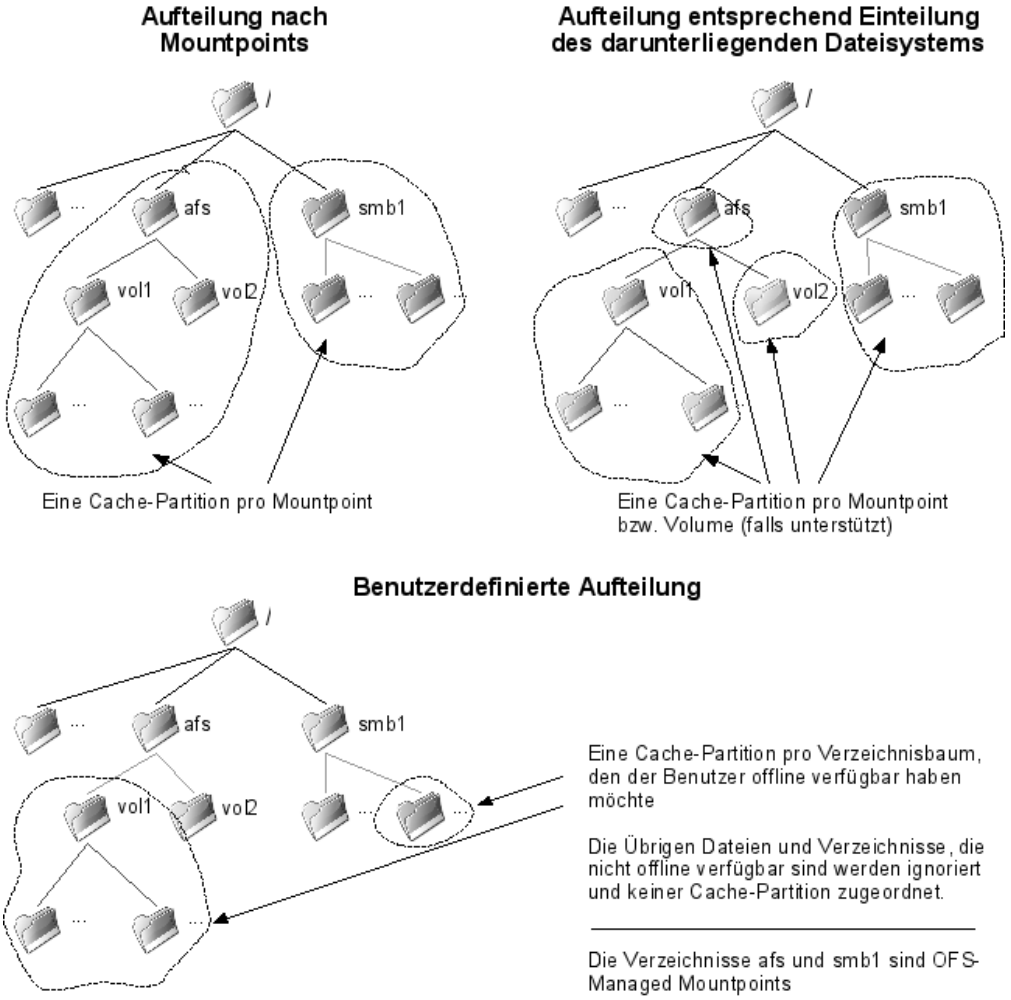

Abbildung 3.1.: Aufteilung des OFS Cache in Partitionen

3.2.3. Aufbau des Client-Caches

Wie bei allen Dateisystemen, die Disconnected Operation unterstützen, benötigt der Client eine lokale Kopie der Dateien auf die er im Falle eines Netzwerkverlustes zurückgreifen kann. Für die Größe des Caches muss von Seiten der Software kein Limit gesetzt werden. Dadurch steht dem Benutzer für die Dateien, die er offline verfügbar haben möchte, die gesamte Kapazität der Festplatte zur Verfügung.

Um die Reintegration zu erleichtern wird pro Cache-Partition ein Logfile geführt, in dem alle Änderungen im Cache notiert werden. Dies entspricht dem CML unter Coda (Abschnitt: 2.6.4). Es ist sehr einfach realisierbar, da sämtliche Dateizugriffe ohnehin „durch“ das OFS gehen. Ausserdem ist es sinnvoll einen Mechanismus einzuführen, der dieses Logfile optimiert und unnötige Einträge entfernt. So heben sich beispielsweise Operationen zum Anlegen und Löschen der selben Datei gegenseitig auf.

3.2.4. Aktualität des Caches

Wenn der Benutzer die Verbindung vom Netzwerk trennt, müssen alle Dateien, die synchronisiert werden sollen, aktuell sein. Da das Dateisystem nicht vorhersehen kann, wann der Benutzer die Netzwerkverbindung löst, müssen die Dateien zu jedem Zeitpunkt aktuell sein.

Als Ansatzpunkt dient das Konzept von Coda. Alle 10 Min. werden die Callback Breaks überprüft und geänderte Dateien neu heruntergeladen. Dadurch sind die Dateien im Cache höchstens 10 Min. alt, wenn die Verbindung getrennt wird. Sollte dies nicht genügen, kann der Benutzer die Synchronisation jederzeit per Hand ausführen.

Falls ein Netzwerkdateisystem verwendet wird, das keine Callback Breaks unterstützt, wie beispielsweise SMB, muss der gesamte Verzeichnisbaum nach Änderungen durchsucht werden. Da dies sehr lange dauert und das Netzwerk strapaziert, wird in diesem

Fall auf die automatische Synchronisation verzichtet und nur auf Wunsch des Benutzers abgeglichen.

Erst bei genauerer Betrachtung wird sich zeigen, ob es einfach möglich ist, die Callback Breaks von AFS zu erkennen. Ist dies nicht möglich, muss u.U. auf die automatische Synchronisation komplett verzichtet werden.

3.2.5. Online- und Offlinebetrieb

Onlinemodus

Jedes Cache-Segment kann sich im Zustand *Online* oder *Offline* befinden. Im *Online* Zustand erfolgt jeder Zugriff direkt auf das darunterliegende Netzwerk-Dateisystem. Gleichzeitig wird die entsprechende Datei im Cache, falls notwendig, aktualisiert.

Wird ein Lesezugriff ausgeführt, überprüft OFS zunächst ob die Datei im Cache noch aktuell ist. Dies geschieht je nach Dateisystem, indem es überprüft, ob ein Callback-Break vorhanden ist bzw. indem es Timestamp und Größe vergleicht. Ist die Datei nicht mehr aktuell oder noch gar nicht im Cache, lädt es diese vom Server. Dies geschieht im Hintergrund, sodass der Zugriff selbst dadurch nicht beeinträchtigt wird.

Schreibzugriffe werden gleichzeitig an das darunterliegende Dateisystem und den Cache weitergegeben.

Offlinemodus

Besteht keine Verbindung zum Server, befindet sich die Cache-Partition im *Offline* Zustand. Hier werden alle Zugriffe, sowohl lesend, als auch schreibend, nur an den Cache geleitet. Gleichzeitig werden diese mitgeschrieben um die Reintegration zu erleichtern.

Erkennung von Online- und Offlinebetrieb

Die Erkennung von Online- und Offlinebetrieb ist ein komplexes Thema und wurde bei den Windows Offlinefiles sehr schlecht umgesetzt. Wird eine Netzwerkverbindung getrennt, bedeutet dies nicht unbedingt, dass alle Cache-Partitions in den offline Zustand wechseln müssen, denn es könnten noch andere Netzwerkverbindungen bestehen. Umgekehrt könnte der Benutzer sein Notebook auch in einem anderen Netzwerk betreiben. Es sind also nicht unbedingt alle Cache-Partitions online, wenn eine Netzwerkverbindung besteht. Schließlich könnte der Offline Zustand auch dadurch hervorgerufen werden, dass der Server vom Netzwerk genommen wird.

Um diese Szenarien unter einen Hut zu bringen, soll eine Kombination aus mehreren Mechanismen verwendet werden:

Prüfen der Verbindung In einem Abstand von 10 Min. überprüft OFS im Hintergrund alle OFS-managed Mountpoints. Kann die Verbindung zu einem Server nicht mehr hergestellt werden, wird das Dateisystem getrennt und alle Cache-Segments, die sich darauf befinden, wechseln in den Offline Zustand. Kann umgekehrt eine getrennte Verbindung erneut hergestellt werden, wird das Dateisystem eingehängt und die Reintegration für alle darauf befindlichen Cache-Segements gestartet.

Zugriffsfehler Schlägt ein Zugriff mit einem Timeout fehl, geht OFS davon aus, dass der Server nicht mehr verfügbar ist. Das entsprechende Cache-Segment wechselt daraufhin in den Offline Zustand. Gleichzeitig wird das darunterliegende Dateisystem aus dem OFS-managed Mountpoint ausgehängt.

Überprüfung bei Verbindungswechsel Das OFS bekommt eine Nachricht, wenn eine neue Netzwerkverbindung vorhanden ist, also wenn der Benutzer beispielsweise

das Netzwerkkabel einsteckt oder eine Wireless LAN Verbindung aufbaut. Dies kann mit dem Network Manager realisiert werden, der Bestandteil vieler aktueller Distributionen ist [27]. Wird eine neue Netzwerkverbindung erkannt, überprüft OFS alle OFS-managed Mountpoints und bindet verfügbare Dateisysteme ein. Die darauf liegenden Cache-Partitions werden anschließend reintegriert. Das Umgekehrte geschieht, wenn eine Netzwerkverbindung getrennt wird.

3.2.6. Reintegration

Markiert der Benutzer einen Verzeichnisbaum als offline verfügbar, wird dieser sofort komplett in den Client Cache kopiert. Wechselt ein Cache-Segment vom Offline- in den Online-Zustand startet die Reintegration. Erst wenn diese abgeschlossen ist, wechselt die Cache-Partition endgültig in den Online Zustand.

Die Reintegration selbst erfolgt in zwei Schritten: Zuerst wird das lokale Änderungsprotokoll abgearbeitet und die Änderungen auf dem Server durchgeführt. Bei jeder Datei wird überprüft, ob sich diese auch auf dem Server verändert hat. Wenn sich Timestamp oder Dateigröße verändert haben wird davon ausgegangen, dass die Datei geändert wurde. Tritt ein Konflikt auf, wird die Datei oder das Verzeichnis nicht abgeglichen und muss vom Benutzer per Hand bereinigt werden. Folgende Situationen können automatisch behoben werden.

Ein auf dem Client neu angelegtes Verzeichnis existiert bereits auf dem Server Dies wird ignoriert. Durch die weitere Reintegration werden dadurch automatisch die Inhalte zusammengefügt.

Ein Verzeichnis wurde auf dem Client verändert Falls sich nur die Struktur der Dateien und Unterverzeichnisse geändert hat, wird diese Operation ignoriert. Wurden auf beiden Seiten die Attribute verändert, tritt ein nicht lösbarer Konflikt auf.

Ein Verzeichnis wurde auf dem Client gelöscht Existiert das Verzeichnis auf dem

Server nicht mehr, wird die Operation ignoriert. Andernfalls wird überprüft, ob es eine oder mehrere Dateien oder Verzeichnisse unterhalb dieses Verzeichnisses gibt, die auf dem Server geändert wurden. Ist dies der Fall gibt es einen nicht auflösbaren Konflikt. Wurden keine Dateien verändert, wird die Aktion durchgeführt und das Verzeichnis gelöscht.

Alle erfolgreich durchgeführten Aktionen werden aus dem Protokoll gelöscht.

Nach dem ersten Schritt sind alle lokalen Änderungen auf dem Server durchgeführt worden. Die zweite Phase sorgt dafür, dass auch alle auf dem Server geänderten Dateien im Cache aktuell sind. Hierbei werden sämtliche Dateien und Verzeichnisse auf dem Server auf Änderungen überprüft und diese lokal durchgeführt. Dies geschieht, wie bei Windows Offlinefiles (Siehe Abschnitt 2.4.3), durch Verglich der Modifikationszeit mit der letzten Synchronisation-Zeit.

Konflikte, die während Phase 1 aufgetreten sind, werden vom Benutzer immer per Hand aufgelöst. Erst danach wechselt das Cache-Segment in den Online Zustand. Bei Dateisystemen, die Callback Breaks nicht unterstützen, sind diese beiden Phasen der normale Vorgang, der bei jeder Synchronisation ausgeführt werden muss.

3.3. Bestandteile und Umsetzung

3.3.1. Anordnung im System

Das OFS ist zwischen dem VFS und dem darunterliegenden Netzwerkdateisystem angeordnet. Sämtliche Zugriffe gehen *durch OFS hindurch*. Coda und InterMezzo teilen das Dateisystem in ein Kernel-Modul und einen User-Space Prozess ein. Zwar läuft der Prozess ausserhalb des Kernels geringfüging langsamer, er ist jedoch wesentlich einfacher zu entwickeln und zu debuggen.

Diese Trennung ist zwar aus Gründen der Pflege, Flexibilität und Erweiterbarkeit sinnvoll, in unserem Fall jedoch vermutlich nicht umsetzbar. Da sämtliche Zugriffe direkt an das Netzwerkdateisystem weitergegeben werden müssen und auch das Cache-Management direkten Zugriff auf das darunterliegende Dateisystem benötigt, gibt es keinen Bestandteil der ausserhalb des Kernels Sinn macht (Abb.: 3.2). Bei der technischen Planung sollte dieses Thema auf jeden Fall noch einmal genauer beleuchtet werden und wenn möglich trotzdem eine Aufteilung geschehen.

3.3.2. Benutzeroberfläche

In jedem Fall muss allerdings eine Schnittstelle zwischen Kernel und Userspace geschaffen werden, um die Kommunikation mit dem Benutzer zu ermöglichen. Die Benutzeroberfläche besteht zum einen aus dem OFS UI-Daemon. Dieser überwacht den Status des OFS Kernels und wird benachrichtigt, sobald ein Konflikt vorliegt, der vom Benutzer bearbeitet werden muss. Die eigentliche Benutzeroberfläche kann eine beliebige Software sein, die mit dem UI-Daemon kommuniziert. Denkbar sind beispielsweise Applets für das Gnome Panel oder KDE. (Abb.: 3.2)

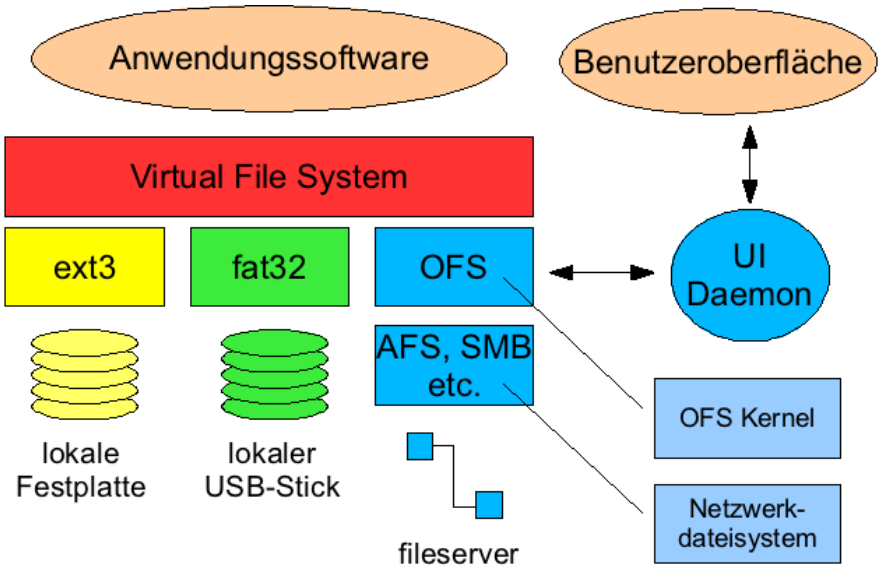

Abbildung 3.2.: Anordnung der OFS Komponenten im System

Kapitel 4.

Ergebnisse

Das Thema *Disconnected Operation* scheint nicht sehr weit verbreitet zu sein. Die Betrachtung der Dateisysteme zeigt, dass offensichtlich wenig Bedarf für Offline Dateisysteme existiert. Aus Gesprächen, die ich während der Zeit meiner Diplomarbeit mit Studenten und Berufstätigen in der EDV und anderen Branchen geführt habe, ist hervorgegangen, dass sehr viele nicht einmal von der Existenz der Windows Offlinefiles wissen. Es ist zwar grundlegendes Interesse an der Thematik vorhanden, jedoch begnügen sich diejenigen, die mobil arbeiten möchten, meistens mit einfachen Synchronisationstools wie SyncToy (Abschnitt 2.4.8) oder kopieren die Dateien einfach per Hand. Software Entwickler verwenden ohnehin Subversion (SVN) oder CVS.

Da der allgemeine Trend zum mobilen Arbeiten geht, ist davon auszugehen, dass auch das Thema Datensynchronisierung immer interessanter wird. Die beste Unterstützung für Mobilität bietet bisher Microsoft. Dazu zählen vor allem Tablet PC Anwendungen. Auch auf das Thema Datensynchronisation legt Microsoft immer mehr Wert. Die Implementierung der Offlinefiles in Windows XP ist zwar unsauber, allerdings wurde diese in Windows Vista verbessert.

Auch wenn unter Linux bereits viel Entwicklungsaufwand- und zeit ins Land gegangen

sind, gibt es dennoch keine brauchbaren offlinefähigen Dateisysteme. Hervorzuheben ist Coda (Abschnitt 2.6), das die längste Entwicklungszeit hinter sich hat. Leider ist dies, wahrscheinlich wegen fehlender Nachfrage, noch immer nicht ausgereift.

Es ist geplant, einen Prototypen für das in Kapitel 3 konzeptionierte Offlinefilesystem zu einem späteren Zeitpunkt zu implementieren. Dieses soll unter einer Open-Source Lizenz freigegeben werden.

Anhang A.

GNU Free Documentation License

Version 1.2, November 2002

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document „free“ in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of „copyleft“, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

Applicability and definitions

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The „Document“, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as „you“. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A „Modified Version“ of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A „Secondary Section“ is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The „Invariant Sections“ are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The „Cover Texts“ are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A „Transparent“ copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a varie-

ty of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not „Transparent“ is called „Opaque“.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The „Title Page“ means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, „Title Page“ means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section „Entitled XYZ“ means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as „Acknowledgements“, „Dedications“, „Endorsements“, or „History“.) To „Preserve the Title“ of such a section when you modify the Document means that it remains a section „Entitled XYZ“ according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

Verbatim copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

Copying in quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's

license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled „History“, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled „History“ in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the „History“ section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled „Acknowledgements“ or „Dedications“, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled „Endorsements“. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled „Endorsements“ or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled „Endorsements“, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

Combining documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled „History“ in the various original documents, forming one section Entitled „History“; likewise combine any sections Entitled „Acknowledgements“, and any sections Entitled „Dedications“. You must delete all sections Entitled „Endorsements“.

Collections of documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

Aggregation with independent works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an ägggregateïf the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warrany Disclaimers, provided that you al-

so include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled „Acknowledgements“, „Dedications“, or „History“, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Future revisions of this License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License „or any later version“ applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Addendum: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled „GNU Free Documentation License“.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the „with...Texts.“ line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Glossar

Dateisystem (filesystem) Ein Ordnungs- und Zugriffssystem für Dateien. Das Dateisystem verwaltet die physikalische Speicherung von Dateien beispielsweise auf einer Festplatte und bringt diese in eine Form (Meist Baumstruktur), in der sie vom Betriebssystem genutzt werden kann.

Disconnected Operation Netzwerk-Dateisysteme, die disconnected Operation unterstützen, ermöglichen es dem Benutzer, auch dann mit gemeinsam genutzten Dateien zu arbeiten, wenn die Verbindung zum Dateiserver unterbrochen wurde.

Journaling-Dateisystem „Ein Dateisystem, welches alle Änderungen vor dem eigentlichen Schreiben in einem dafür reservierten Speicherbereich, dem Journal, aufzeichnet.“[49] Erst wenn die *commit* Aktion ausgeführt wird, werden die Daten geschrieben. Dadurch werden Inkonsistenzen bei einem Systemabsturz verhindert.

Kerberos „Kerberos bietet sichere und einheitliche Authentifizierung in einem ungesicherten TCP/IP-Netzwerk auf sicheren Hostrechnern. Die Authentifizierung übernimmt eine vertrauenswürdige dritte Partei. Diese dritte Partei ist ein besonders geschützter Kerberos-5-Netzwerkdienst.“[50]

Network Basic Input Output System (NetBIOS) NetBIOS wurde im Auftrag von IBM entwickelt. Es sollte eine Abstraktionsebene für den Netzwerkzugriff dar-

stellen, ähnlich dem Basic Input Output System (BIOS), das eine Abstraktionsebene zum Zugriff auf die Hardware bildet. [51]

Netzwerkdateisystem Ein Dateisystem, das als Client für ein Netzwerkprotokoll agiert und seine Datenbasis auf einem entfernten Server hat.

Offlinefilesystem Ein Offlinefilesystem ist ein Dateisystem, das disconnected operation unterstützt. Es ermöglicht dem Benutzer, auch dann mit gemeinsam genutzten Dateien zu arbeiten, wenn keine Verbindung zum Dateiserver besteht.

Replikation „Replikation oder Replizierung bezeichnet die mehrfache Speicherung von Daten an typischerweise unterschiedlichen Standorten.“[53] Das Ziel vom Replikationen ist es meistens, Server- oder Netzwerkausfälle zu überbrücken oder die Last auf mehrere Rechner zu verteilen.

Request for Comments „Die Requests for Comments (kurz RFC; zu deutsch Forderung nach Kommentaren) sind eine Reihe von technischen und organisatorischen Dokumenten des RFC-Editors zum Internet, die am 7. April 1969 begonnen wurden. Bei der ersten Veröffentlichung noch im ursprünglichen Wortsinne zur Diskussion gestellt, behalten RFC auch dann ihren Namen, wenn sie sich durch allgemeine Akzeptanz und Gebrauch zum Standard entwickelt haben.“[54]

Transaktion Eine Transaktion ist eine Folge zusammengehöriger Operationen. Der Vorteil von Transaktionskonzepten liegt darin, dass die Operationen vor dem endgültigen Festschreiben problemlos rückgängig gemacht werden können.

Versionsverwaltung Versionsverwaltung wird häufig in der Softwareentwicklung eingesetzt. Mehrere Entwickler arbeiten gleichzeitig an einer lokalen Kopie der Software. Nach jedem Entwicklungsabschnitt wird diese in die gemeinsame Ablage übertragen (commit). Der Stand der Software zwischen den commit Vorgän-

gen kann jederzeit abgefragt werden. Gängige Versionsverwaltungssysteme sind CVS und SVN.

Zustandsloses Protokoll Alle Anfragen werden unabhängig voneinander bearbeitet, auch wenn diese vom selben Absender stammen. Es werden keine Sitzungs- oder Statusinformationen über die Clients gespeichert. [55]

Abkürzungen

CMU Carnegie Mellon University

VFS Virtual File-System

AFS Andrew File-System

ASR Application-specific resolver

CML Client Modification Log

CSC Client Side Cache

CFS Cluster File Systems, Inc

KML Kernel Modification Log

OFS Ohm File-System

DFS Microsoft Distributed File System

NFS Network File System

SMB Server Message Block

SML Synchronisation Modification Log

TCP Transmission Control Protocol

CIFS Common Internet File System

ACL Access Control List

UDP User Datagram Protocol

IETF Internet Engineering Task Force

NIS Network Information Service

CVS Concurrent Versions System

SVN Subversion

CGI Common Gateway Interface

SSH Secure Shell

DHCP Dynamic Host Configuration Protocol

GPL General Public License

LGPL Lesser General Public License

Literaturverzeichnis

[1] ALLCHIN, Jim: *Windows Vista Team Blog : Offline Files*. http://windowsvistablog.com/blogs/windowsvista/archive/2007/01/29/working-with-offline-files.aspx, Abruf: 06.06.2007

[2] BRAAM, Peter ; BARON, Robert ; HARKES, Jan ; SCHNIEDER, Marc: *The Coda HOWTO*

[3] BRAAM, Peter J.: *Re: [ANNOUNCE] Lustre Lite 1.0 beta 1*. Linux Filesystem Development Mailinglist. http://osdir.com/ml/file-systems/2003-03/msg00035.html

[4] BRAAM, Peter J.: *The Coda Distributed File System*. http://www.coda.cs.cmu.edu/ljpaper/lj.html, Abruf: 15.05.2007

[5] BRAAM, Peter J.: The Coda Distributed File System. In: *LINUX Journal* 6 (1998), S. 46–51

[6] BRAAM, Peter J.: *Intermezzo: File Synchronization with Intersync*. 1999

[7] BRAAM, Peter J. ; CALLAHAN, Michael ; SCHWAN, Phil: *The InterMezzo File System*. 1999

[8] CLUSTER FILE SYSTEMS, Inc. w.: *InterMezzo File System Home*. http://www.inter-mezzo.org, Abruf: 09.07.2007

[9] CLUSTER FILE SYSTEMS, INC: *Cluster File Systems, Inc. Website*. http://www.clusterfs.com, Abruf: 13.06.2007

[10] CLUSTER FILE SYSTEMS, INC.: *Lustre: A Scalable, High-Performance File System*. http://www.clusterfs.com/resources.html

[11] CLUSTER FILE SYSTEMS, INC.: *lustre Datasheet*. http://www.clusterfs.com/resources.html

[12] CLUSTER FILE SYSTEMS, INC.: *Lustre Features Roadmap*. http://www.clusterfs.com/roadmap.html, Abruf: 13.06.2007

[13] TEAM, Coda D. (Hrsg.): *Coda File System Website*. http://www.coda.cs.cmu.edu/, Abruf: 15.05.2007

[14] COLLEGE OF AGRICULTURAL SCIENCES ITC: *How To Work with Offline Files*. http://ict.cas.psu.edu/Training/HowTo/ENComputers/offlinefiles.htm, Abruf: 23.05.2007

[15] DANIEL KOBRAS, Joachim K.: Eigenbrötler - AFS im Vergleich mit SMB und NFS. In: *iX Magazin für professionelle Informationstechnik* 6 (2007), Juni, S. 116–118

[16] *GNU General Public License, version 2*. http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

[17] *GNU Lesser General Public License, version 2.1*. http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html

[18] *GNU General Public License, version 3*. http://www.gnu.org/copyleft/gpl.html

[19] HAGEN, Bill von: *Modern Distributed Filesystems For Linux: An Introduction*. Version: August 2002. http://linuxplanet.com/linuxplanet/reports/4361/1/, Abruf: 09.07.2007

[20] HAGEN, Bill von: *Using the InterMezzo Distributed Filesystem*. Version: August 2002. http://www.linuxplanet.com/linuxplanet/reports/4368/1/, Abruf: 13.06.2007

[21] *The Internet Engineering Task Force Website*. www.ietf.org

[22] KISTLER, James J. ; SATYANARAYANAN, M.: Disconnected operation in the Coda File System. In: *ACM Trans. Comput. Syst.* 10 (1992), Nr. 1, S. 3–25. http://dx.doi.org/http://doi.acm.org/10.1145/146941.146942. – DOI http://doi.acm.org/10.1145/146941.146942. – ISSN 0734–2071

[23] GROUP, Waiko Linux U. (Hrsg.): *Lustre - Waiko Linux Users Group*. http://www.wlug.org.nz/Lustre, Abruf: 13.06.2007

[24] MICROSOFT: *What's New in Offline Files for Windows Vista*. http://technet2.microsoft.com/WindowsVista/en/library/bb819260-0fdc-4003-bc23-04beac2108bd1033.mspx?mfr=true, Abruf: 12.06.2007

[25] MICROSOFT: *WIndows Vista Help: Understandung offline files*. http://windowshelp.microsoft.com/Windows/en-US/Help/93a550df-34cd-4497-85d0-8732602f59591033.mspx, Abruf: 06.06.2007

[26] *Microsoft Distributed File System Demonstation*. http://www.microsoft.com/windowsserver2003/evaluation/demos/dfs.html

[27] *Network Manager*. http://www.gnome.org/projects/NetworkManager/

[28] OPENAFS: *openAFS Administration Guide*. http://www.openafs.org/pages/doc/AdminGuide/auagd000.htm

[29] *[OpenAFS] Windows Offline folders or Briefcase and AFS?* http:

//www.openafs.org/pipermail/openafs-info/2004-August/014588.html

[30] *OpenOffice.org Website*. www.openoffice.org

[31] *Open Source Initiative OSI - IBM Public License Version 1.0:Licensing*. http://www.opensource.org/licenses/ibmpl.php

[32] SAITO, Yasushi ; SHAPIRO, Marc: Optimistic Relication. In: *ACM Computing Surveys* 37 (2005), March, Nr. 1, S. 42–81

[33] *Samba Website*. www.samba.org

[34] SATYANARAYANAN, M.: The evolution of Coda. In: *ACM Trans. Comput. Syst.* 20 (2002), Nr. 2, S. 85–124. http://dx.doi.org/http://doi.acm.org/10.1145/507052.507053. – DOI http://doi.acm.org/10.1145/507052.507053. – ISSN 0734–2071

[35] SATYANARAYANAN, M. ; EBLING, Maria R. ; RAIFF, Joshua ; BRAAM, Peter J. ; HARKES, Jan: *Coda File System User and System Administrators Manual*. CMU, 2000. http://www.coda.cs.cmu.edu/doc/html/manual/index.html

[36] SATYANARAYANAN, M. ; KISTLER, James J. ; MUMMERT, Lily B. ; EBLING, Maria R. ; KUMAR, Puneet ; LU, Qi: Experience with Disconnected Operation in a Mobile Computing Environment. In: *Proceedings of the 1993 USENIX Syposiom on Mobile and Location-Independent Computing* (1993)

[37] SHEPLER, S: *RFC 2624 - NFS Version 4 Design Considerations*. http://www.faqs.org/rfcs/rfc2624.html, Abruf: 31.08.2007

[38] SHEPLER, S ; CALLAGHAN, B ; ROBINSON, D ; THURLOW, R ; BEAME, C ; EISLER, M ; NOVECK, D: *RFC 3530 -Network File System (NFS) version 4 Protocol*. http://www.faqs.org/rfcs/rfc3530.html, Abruf: 31.08.2007

[39] TEAM, Coda D.: *Coda Frequently asked Questions*. http://www.coda.cs.cmu.edu/misc/, Abruf: 05.06.2007

[40] TRIDGELL, Andrew ; MACKERRAS, Paul: The rsync algorithm / Australian National University. Version: 1998. http://samba.anu.edu.au/rsync/tech_report/. Canberra, ACT 0200, Australia, 1998. – Forschungsbericht

[41] *Unison File Synchronizer*. http://www.cis.upenn.edu/~bcpierce/unison/

[42] VIRK, Navjot: *The Filing Cabinet : Offline Files in Windows Vista*. https://blogs.technet.com/filecab/archive/2006/07/11/441131.aspx, Abruf: 06.06.2007

[43] *Wikipedia: Access Control List*. http://de.wikipedia.org/wiki/Access_Control_List

[44] *Wikpedia: Coda (Dateisystem)*. http://de.wikipedia.org/wiki/Coda_%28Dateisystem%29

[45] *Wikipedia: Dateisystem*. http://de.wikipedia.org/wiki/Dateisystem

[46] *Wikipedia: Dynamic Host Configuration Protocol*. http://de.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol

[47] *Wikipedia: Hash-Funktion*. http://de.wikipedia.org/wiki/Hash-Funktion

[48] *Wikipedia: InterMezzo (file system)*. http://en.wikipedia.org/wiki/InterMezzo_%28file_system%29

[49] *Wikipedia: Journaling-Dateisystem*. http://de.wikipedia.org/wiki/Journaling-Dateisystem

[50] *Wikipedia: Kerberos (Informatik)*. http://de.wikipedia.org/wiki/

Kerberos_%28Informatik%29

[51] *Wikipedia: NetBIOS*. http://de.wikipedia.org/wiki/Netbios

[52] *Wikipedia: Prüfsumme*. http://de.wikipedia.org/wiki/Checksumme

[53] *Wikipedia: Replikation (Datenverarbeitung)*. http://de.wikipedia.org/wiki/Replikation_%28Datenverarbeitung%29

[54] *Wikipedia: Request for Comments*. http://de.wikipedia.org/wiki/Request_for_Comments

[55] *Wikipedia: Zustandslosigkeit*. http://de.wikipedia.org/wiki/Zustandslosigkeit