Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Thesis Outline

Lutz Behnke

Kooperativer Speicher: Schwächen und Gegenmaßnahmen

Fakultät Technik und Informatik
Studiendepartment Informatik

Lutz Behnke

Thema der Thesis Outline

Kooperativer Speicher: Schwächen und Gegenmaßnahmen

Stichworte

Netzwerk Persistenz Storage

Kurzzusammenfassung

In mehreren Bereichen der IT-Technik sind heute Speichersysteme nicht mehr geeignet die Anforderungen an Flexibilität, Skalierbarkeit und Stabilität gegen Fehler zu erfüllen. Kooperative Speicher scheinen ein Ausweg aus der Konfigurations-Falle zu sein, die sich aus immer komplexeren Lösungen ergibt. Im folgenden wird ein Ansatz, der zukünftig der Gegenstand der Masterarbeit des Autors werden soll, in Kontext gestellt und notwendige Arbeiten und Untersuchungen diskutiert.

Inhaltsverzeichnis

1 Einleitung

1.1 Das Problem

Aus einer Reihe von Bereichen der IT kommen heute Forderungen nach verbesserten Storage-Systemen. Vor allem Konzepte wie *Ubiqutous Computing* mit seiner Vielzahl von allgegenwärtigen, intelligenten Geräten, *Grid Computing* mit Domänen übergreifenden Anwendungen und *Cluster Computing* mit einer enormen Anzahl an Rechner-Knoten verlangen nach Verbesserungen in den Punkten

- Ad-Hoc Konfiguration/Discovery
- Skalierbarkeit
- (scheinbarer) Stabilität
- Organisatorische Transparenz

Kein Mensch, egal ob Fachmann oder Laie kann die, sich ständig verändernde, Zusammenstellung von Geräten und Kapazitäten in den obigen Bereichen zeitnah konfigurieren.

Daher benötigen alle Elemente die Fähigkeit, sich selbständig nach gewissen Regeln in ihrer, sich u.U. ändernden, Umgebung zu konfigurieren.

Neben der Möglichkeit, eine arbiträre Anzahl von Elementen zu unterstützen[1] stellt vor allem die, sich ständig ändernde, Umgebung und Netzzusammenstellung eine hohe Anforderung an ein Speichersystem. Bei begrenzter Reichweite von Bluetooth und WLAN muss ein System trotz ständig erscheinenden und verschwindenen Knoten stabil die Datenübertragung fortsetzen.

Projekt Oxigen(Oxigen 2004) beschreibt einige der Anforderungen aus Sicht des Ubiqutous Computing.

1.2 Lösungsansatz: Kooperative Speicherung

Während der Analyse von Fehlerfällen in Systemen zur Datenspeicherung haben mehrere Autoren (z.B. (Goh u. a. 2003),(Mazières und Shasha 2002))die Trennung der Sicht auf die Datenspeicherung aus Sicht des Anwenders und aus Sicht des Betreibers des Datenspeichers untersucht. Sind Anwender und Betreiber/Besitzer des Speichers unterschiedlich, so verliert in der Regel der Anwender einen Teil der Kontrolle über die Art und Weise in der seine Daten abgelegt sind. Daraus ergibt sich folgende Definition für Kooperative Speicherung:

> Jede Form der Datenspeicherung, bei welcher der Anwender keine oder unvollständige Kontrolle über das Speichermedium ausübt

Dabei beschreibt “kooperativ” nicht die gesellschaftliche oder rechtliche Relation von Dateneigentümer und Betreiber des Speichers. Sie können durch externe Regelungen oder Verträge in einem Zwangsverhältnis stehen. Einzig der Mangel an vollständiger Kontrolle über die Art und Weise der Speicherung durch den Anwender dient als Abgrenzung ob eine kooperative Speicherung vorliegt oder nicht.

Kooperative Speicherung kommt in einer Reihe von Anwendungsfällen aus unterschiedlichen Bereichen vor:

Ad-Hoc/Mobile Communities Die Anzahl der, an Datenübertragung und -speicherung beteiligten, Geräte ist groß und ändert sich durch die Bewegung der Besitzer ständig, so das eine Konfiguration durch einen Menschen unmöglich ist und automatisch geschehen muss. Dies ist Gegenstand der Betrachtung durch das Oxigen Projekt des MIT und auch für unseren Ferienclub interessant.

Eine besondere Variante dieses Anwendungsfalles ist der Einsatz von Geräten ganz

[1] Es gibt in Deutschland mehr als 80 Millionen Einwohner und im Juni 2005 74.1 Millionen Handys. Und das obwohl 30% aller Deutschen bei einer Umfrage sagten das sie gar kein Handy haben wollen

ohne Benutzerschnittstelle wie dem “Personal Data Stick”[2] wie er vor einigen Jahren von Hewlett-Packard vorgestellt wurde.

Anwendungen mit Mandatory Access Controls (MAC) In Bereichen in denen MACs, wie etwa nach dem Bell/LaPadula Modell (Bell und LaPadula 1976), z.B im militärischen Umfeld, eingesetzt werden, muss der Administrator eines klassischen Systems mindestens so hohe Freigaben haben, wie der Anwender mit der höchsten Freigabe. Da dies aus praktischen Gründen selten möglich ist (Der General darf alles lesen, aber nicht der Gefreite, der den Computer wartet), kann die Anforderung als Form der kooperativen Speicherung betrachtet werden.

Aus diesen Anwendungsbereich ergibt sich deutlich die Anforderung nach Ende-zu-Ende Verschlüsselung (End-to-End Encryption, E2EE). Nur mit Hilfe von E2EE lassen sich MACs auch in Umgebungen mit kooperativer Speicherung durchsetzen.

Outsourcing Dies ist wohl der zur Zeit kommerziell am relevanteste Bereich zum Thema kooperative Speicherung. Denn egal wie streng der Vertrag zwischen den Partnern ist, so ist doch die obige Definition erfüllt. Und ein System das Sicherheit und Stabilität liefert und die Konfigurationsaufgaben minimiert oder auf den Anbieter des Speicherplatzes abwälzt ist kommerziell relevant. Für den Anbieter des Speicherdienstes ist es natürlich ebenso interessant minimale Kostenaufwände durch die Konfiguration seiner Systeme zu haben und durch eine dokumentierbare Mandantenfähigkeit die Daten seiner unterschiedlichen Kunden von einander zu trennen.

High Performance Computing (HPC) während in diesem Bereich sowohl die Daten als auch die Speichergeräte unter der Kontrolle einer Organisation stehen, ergibt sich aus der enormen Anzahl der beteiligten Komponenten ein ähnliches Problem wie in Ubiqutous Umgebungen: kein Mensch kann mehr die Konfiguration aller Einzelteile leisten. Ansätze wie das GoogleFS (Ghemawat u. a. 2003) und Lustre (CFS 2002) versuchen dem Problem durch die konsequente Trennung von Daten und Metadaten Herr zu werden. Grundlage von beiden Systemen sind wenige ausgezeichnete Meta-Server bei denen sich die eigentlichen Datenserver registrieren und somit ein limitiertes Discovery der Ressourcen durchgeführt wird.

Grid-Computing Das grundlegende Problem der Entdeckung und Konfiguration von Ressourcen verschlimmert sich natürlich wenn die einzelnen Elemente in unterschiedlichen Organisationen und administrativen Domänen beheimatet sind. Das Grid-Computing verspricht aber gerade den Anwendern über diese Grenzen hinweg Rechen- und Speicherleistung zu liefern. Die heute verfügbaren Ansätze(LNCS 3779

[2] Das Gerät ist im Prinzip ein Bluetooth-Dongle mit Batterie und speichert alle Daten die von gleichartigen Geräten publiziert werden und die vom Anwender in Reichweite getragen werden.

2005) sind entweder nicht in der Lage dies vollständig transparent zu leisten oder wälzen die Arbeit auf den Administrator ab (Beintner und Osius (2005), Kap. 9).

1.3 Anforderungen

Aus der Betrachtungen im letzten Abschnitt stellen sich eine Reihe von Anforderungen an ein System zur kooperativen Speicherung:

Discovery Jede im System beteiligte Komponente muss die von ihr benötigten anderen Komponenten selbstständig finden können oder in einen geeigneten Zustand übergehen wenn dies nicht möglich ist.

Read-Write Systeme auf Basis von Peer-2-Peer Protokollen wie das Cooperative Filesystem (CFS (Dabek u. a. 2001)) bieten nur lesende Operationen. Das Schreiben von Daten in das Speichersystem geschieht in einem Out Of Band Verfahren das nicht Bestandteil des Speichersystems ist.

Ende zu Ende Verschlüsselung

Freshness Garantie Sowohl Änderungen an Dateien als auch auch die Änderung von evtl. vorhandenen Meta-Informationen müssen nach ihrem Ändern durch einen Client auch umgehend bei anderen Clients sichtbar sein und vor allem muss sichergestellt werden das die anderen Clients den Versuch einer Manipulation durch einen Man-in-the-middle Angriff zumindest detektieren können.

Skalierbarkeit Die Gesamtzahl der an das Internet angeschlossenen Geräte ergibt potentiell die Zahl der Anwender des Speichersystems. Daher darf es keinen einzelnen Punkt im gesamten System geben der nicht potentiell mit der Anzahl der Clients mit wachsen könnte.

Stabilität gegen Datenverlust Da eine Gruppe von Zielplattformen billige PDAs und andere mobile Geräte sind, die schon durch ihre Benutzung außerhalb einer sicheren Büroumgebung einem höheren Risiko durch Ausfall, Beschädigung oder Zerstörung ausgesetzt sind, muss das System auch mit dem katastrophalen Versagen jeder einzelnen Komponente zurechtkommen ohne Daten zu verlieren.

Übliche Anforderungen, wie niedrige Latenz für Schreib- wie Lesezugriffe oder auch die einzelne ebenso wie aggregierte Bandbreite müssen zur Erfüllung der anderen Anforderungen in den Hintergrund treten. Sollten sie natürlich derart anwachsen, das mit dem System nicht mehr praktisch gearbeitet werden kann, so kann ein Ansatz ebenso als gescheitert gelten, als wenn die oberen Anforderungen nicht erfüllt werden.

Aus obigen Anforderungen definieren sich auch die Abgrenzungen gegenüber klassischen Speicherformen wie Remote Filesystems oder Datenbank-Managementsystemen:

- klassischen Remote-Dateisysteme wie NFS, CIFS, CODA, AFS, etc. bieten weder entweder keine oder nur eingeschränkte Discovery. Ein gutes Beispiel bietet das Common Internet File System (CIFS) das ein Discovery nur innerhalb einer Broadcast Domäne erlaubt.

- auch skalieren diese Dateisysteme nicht transparent. Zwar erlauben die neueren ein dynamisches Ändern der Dateisystemgröße, aber ein redundantes Auslegen der Server-Hardware um zusätzliche Clients zu unterstützen wird nicht angeboten.

- Das größte Problem aber ist die Server zentrierte Zugriffskontrolle. Wenn dem Betreiber des Servers kein Vertrauen geschenkt wird und ein E2EE notwendig erscheint, dann ist eine Server zentrierte Zugriffskontrolle kontraproduktiv. Im Gegenteil, da viele System nicht standardmäßig Public-Key Kryptographie für die Authentifikation nutzen, sind Single-Sign-On Lösungen nur schwer umzusetzen und der Anwender wird gezwungen sich ein Passwort zu merken das er u.U. im Klartext den Server erreicht (egal ob es verschlüsselt übertragen wird oder nicht). Da Anwender oft die gleichen Passwörter mehrfach benutzen werden so den Betreiber des Servers wertvolle Hilfen beim Angriff auf andere Dienste, die der Anwender rechtens benutzt, gegeben.

- Datenbank-Management-Systeme (DBMS) sind ebenso ungeeignet, da sie a priori eine Struktur der Daten erwarten. Gerade für Medien- und andere unstrukturierte Massendaten ist aber unter Umständen die Struktur nicht bekannt oder nicht für eine Ablage in Datenbanken geeignet.

- Kooperative Speichersysteme sind auf die Ablage von Bulk-Daten fokussiert. Solche Daten werden selbst in modernen DBMS oft außerhalb der eigentlichen Tabellenstruktur in sog. `text` oder BLOb Dateien abgelegt und aus der Tabelle heraus nur referenziert.

1.4 Schwächen existierender Ansätze

Alle existierenden Ansätze erfüllen nur einen Teil der unter 1.3 genannten Anforderungen. Folgender Überblick gibt eine Vorstellung davon für welche Aufgaben es bereits Lösungen gibt:

1.4.1 Cooperative File System (CFS)

Dateisystem auf Basis des Chord(Karger u. a. 2001) Peer-to-Peer Frameworks. Durch seine p2p-Basis ist es sehr robust, und skalierbar. Aus dem grundsätzlichen Ansatz des Projekt Oxigens heraus war zu dem ein Ad-Hoc Verhalten eines der Designziele, da im Projekt von einem schnellen Erscheinen und Verschwinden von Knoten ausgegangen wird.

Allerdings zeigt es seine p2p-Wurzeln auch sehr stark in der Tatsache das es ein Read-Only Dateisystem ist. Ein Schreiben ist innerhalb des Systems nicht möglich, sondern Inhalte müssen durch Mechanismen außerhalb des Systems publiziert werden.

1.4.2 Cluster Dateisysteme: GoogleFS, Lustre

Der Umgang mit gigantischen Datenmengen und die intelligente Verteilung der Arbeitslast sind zentrale Themen für Cluster-Filesysteme. Auch ist in einem System das aus vielen tausend einzelnen Computern besteht stets davon auszugehen das ein Teil der Komponenten nicht korrekt funktioniert.

Erreicht wird diese Funktionalität durch eine Trennung von Daten und Metadaten, so das die Übertragung der Nutzdaten direkt zwischen dedizierten Speicherknoten und dem generierenden Knoten abläuft. Ein Metadaten-Server koordiniert nur und hat ein, um drei bis vier Größenordnungen, geringes Datenvolumen als das Gesamtsystem zu handhaben. Dies macht auch eine Replikation dieser zentralen Komponente möglich, so das eine Redundanz im Falle eines Ausfalls zur Verfügung steht. Die Konsistenz der Nutzdaten wird durch Parität-Informationen dem RAID5 nicht unähnlich erreicht.

Ein Nachteil, der auch in das Blickfeld der Entwickler von Lustre gerutscht ist, ist der Mangel an automatischer Konfiguration. Dies ist nicht zu gravierend für den Anwendungszweck, da fast alle beteiligten Knoten gleichartig sind, so das eine einzelne Konfiguration auf alle Knoten des Systems kopiert werden kann. Dennoch soll dieser Umstand abgestellt werden, um den allgemeinen Konfigurationsaufwand eines High-Performance-Clusters zu reduzieren, der sich aufgrund ständig fallender Hardware-Preise als Kostentreiber einer Installation herausbildet.

Die, meist geschlossene, Umgebung und die Begrenzung des Zugriffs auf den Cluster durch eine sog. Headnode machen einen Schutz von Dateien gegen unberechtigte Zugriffe innerhalb des Systems überflüssig. Daher verfügen die Cluster-Dateisysteme über keine E2EE.

1.4.3 Cooperative Front-Ends: SiRius

In (Goh u. a. 2003) wird die Möglichkeit untersucht, mit einem sicheren Front-End auch Back-Ends, denen nicht vertraut werden kann, zu unterstützen. Dies bedeutet in erster Linie eine Ende-zu-Ende Verschlüsselung, aber liefert auch die Möglichkeit, unterschiedliche Back-Ends zu unterstützen. Dabei werden nicht nur klassische Netzwerkdateisysteme wie NFS oder SMB untersucht, sondern auch Web-basierte Speichersysteme wie Yahoo!.

Ein zentraler Punkt der oben genannten Anforderungen aber, nämlich die automatische Entdeckung von Speicherorten wird bei SiRius nicht betrachtet und auch nicht unterstützt. Auch betrachtet der Ansatz nicht die Frage nach Replikation durch das Backend, sondern er-

wartet einfach das ein angemessenes Backup durch die Betreiber der Back-Ends unterstützt wird.

2 Mein Ansatz

Im folgenden ist Wotan, ein prototypisches System zur Realisierung der unter 1.3 genannten Anforderungen beschrieben.

2.1 Vision

Die grundsätzliche Vision meines Ansatzes beruht auf dem Wunsch, aus User-Sicht, Daten in Anwendungen abzuspeichern und bei Bedarf wieder zurück zu erlangen, ohne das der Anwender sich jemals über Details wie Speicherorte, Backup, volle Platten oder ähnliche alltägliche Schwierigkeiten der Speicherung Gedanken machen muss.

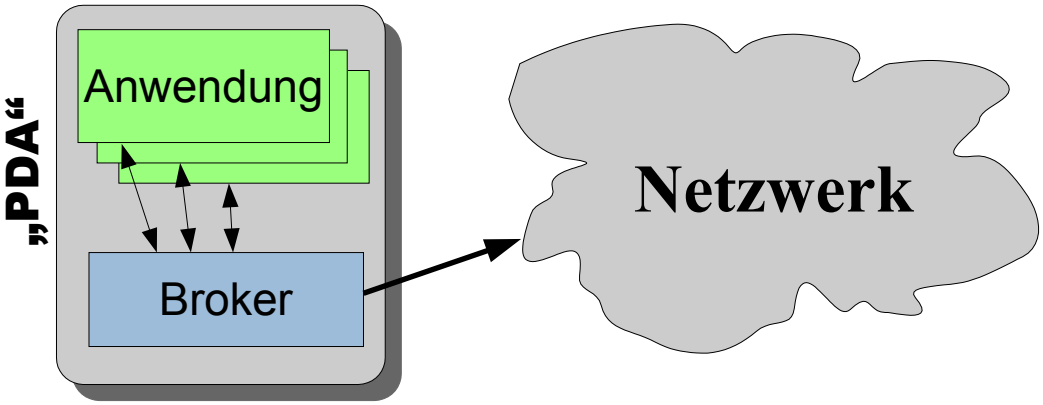

Abbildung 1: Vision

Neben der Frage der Speicherung ist sollte aber auch die Verwaltung der gespeicherten Daten nicht von dem Ort der Speicherung oder einem festgelegten Pfad zum Speicherort den Namen der Datei bestimmen. Daher gibt es in meiner Vision keinerlei Metadaten, Pfade oder Verzeichnisse. Dies sind alle Informationen die nur der Anwender benötigt und daher von einer entsprechenden Software gelöst werden sollte, die auf die Form und Funktionalität der Hardware-Plattform abgestimmt sein soll[3]. Für den Computer ist ein eindeutiger Identifier vollkommen ausreichend um ein Speicherobjekt wieder zu finden.

Damit nicht jede Anwendung auf einem System die notwendige Logik für das Auffinden von Speicherorten und der Verwaltung von Caches beinhalten muss, gibt es auf jedem Computer einen Broker, der die Kommunikation mit der Umwelt des Computers realisiert. Ob

[3] gemäß dem Unix-Dogma “Ein Programm für eine Aufgabe, eine Aufgabe für ein Programm”

dies wirklich als getrennte Anwendung realisiert wird, als Bibliothek, die von Anwendungen geladen werden kann oder letztendlich als Teil des Betriebsystems, ist dabei ein Implementationsdetail.

2.2 API

Eine Applikation nutzt dabei die denkbar simpelste API:

```
FileHandle sendData(byte[] data);
byte[] getData(FileHandle handle);
```

wobei das `FileHandle` ein einfaches, serialisierbares Objekt ist, das leicht lokal gespeichert oder auch über das Netz übertragen werden kann. Für eine reale Implementation muss der genaue Aufbau des FileHandles definiert werden, höchst wahrscheinlich eine einfache Byte-Folge als Ergebnis einer Hash-Funktion.

Was auf den ersten Blick gegenüber normalen File-System APIs auffällt ist das Fehlen von Funktionen zur Manipulation von Meta-Daten ebenso wie das Löschen von Dateien. Dies hat den simplen Grund, das ein Löschen von Dateien grundsätzlichen nicht vorgesehen ist. Und das Meta-Daten von diesem System nicht gespeichert werden[4]. Meta-Daten bedeutet in diesem Zusammenhang auch Datei- oder Ordner-Namen.

2.3 Ziele für einen Prototypen

Während der endgültige Einsatzort für Wotanauch mobile Geräte umfassen soll, bieten diese begrenzten Umgebungen doch eine Menge zusätzlicher Herausforderungen, so das der erste Prototyp auf einer gewöhnlichen Java-SE Virtual Maschine lauffähig sein wird.

Als minimale Anforderungen an den Prototypen ergeben sich folgende Punkte:

- Daten werden abgespeichert
- Vollständige Discovery für Geräte, Ressourcen, Konfigurationen
- End-2-End-Encryption

Dabei werden folgende Annahmen gemacht: Die notwendige Crypto-Implementation wird als vorhanden vorausgesetzt, oder durch den Einsatz von Blind-Algorithmen ersetzt. Und es existiert ein Mechanismus der Discovery von Netzwerk-Ressourcen. Diese werden dem Broker gemeldet, er muss sie aber nicht selbständig entdecken.

Dieser Prototyp wird bis auf die Verschlüsselung als Gegenstand des Projekt im 3. Semester des Master-Studienganges von mir entwickelt.

[4] jedenfalls nicht direkt. Wenn eine Middleware zur Speicherung von Meta-Informationen das System zu Ablage ihrer Daten benutzen möchte ist das eine andere Sache

2.4 Erweiterte Architektur

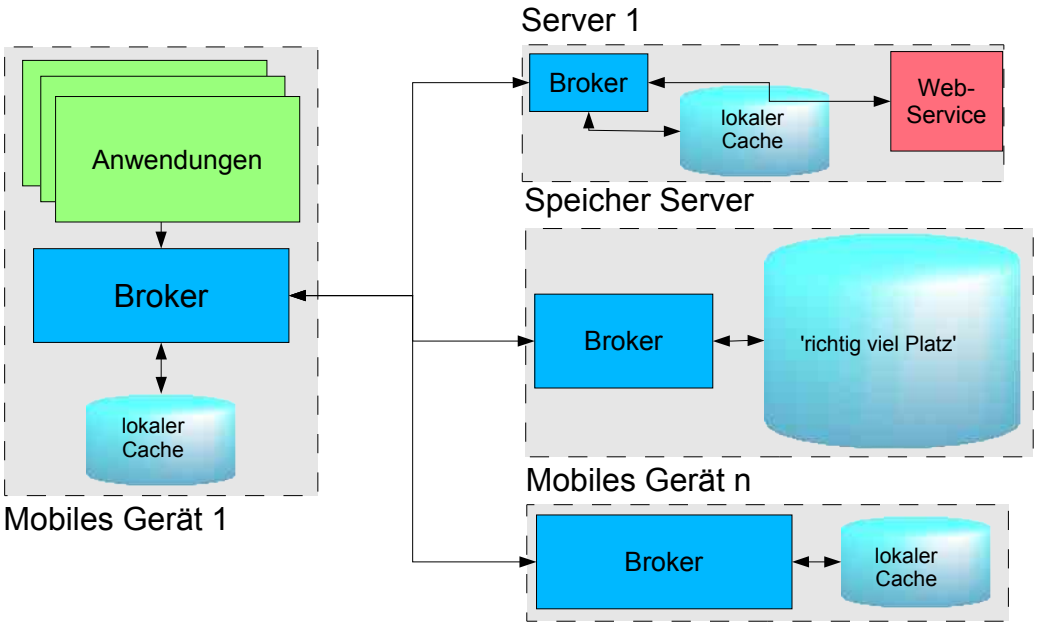

Abbildung 2: Architektur Übersicht

Aus praktischen Überlegungen ergibt sich die Architektur wie in 2 dargestellt für ein realistisches Anwendung-Szenario, in dem sowohl mobile Geräte als auch ortsfeste Geräte auf den gemeinsamen Datenraum zugreifen können. Das Ziel der unterschiedlichen System ist es aus den verfügbaren Daten einen Mehrwert zu schöpfen:

- Die Möglichkeit mehr Daten abzulegen als auf dem eigenen mobilen Gerät Speicherplatz vorhanden ist, ist ein Ziel von Wotan. Dies wird durch die Bereitschaft anderer Besitzer von mobilen Geräten ermöglicht unbenutzten Platz zur Verfügung zu stellen. Die Frage welcher Platz wieder freigegeben werden kann und welche notwendigen Vertrauensverhältnisse, die übergebenden Daten sicher zu verwahren oder wenn möglich an einem permanenteren Platz zu speichern werden während der Master-Arbeit untersucht.

- Natürlich ist eine ausreichende Datenmenge auch in einer großen Anzahl an mobilen Geräten nicht abzulegen, da mobile Geräte vor allem eine erhöhte Gefahr des Verlustes oder der Zerstörung haben als Server in abgeschlossenen Räumen. Daher ist ein integraler Bestandteil des Systems eine Reihe von Servern mit ausreichend dimensionierten Speichermedien um momentan nicht benötigte Dateien aufzunehmen. Die

durch diese Architektur möglichen makroskopischen Antwortzeiten nimmt das Design des Systems billigend in Kauf.

Diese Server sind in einem Internet-weiten Betrieb von Wotandurchaus als Dienst von kommerziellen Anbietern zu betreiben. Eine Untersuchung von derartigen Geschäftsmodellen wird nicht Teil der Master-Thesis sein.

- Die notwendigen Zugangsberechtigungen vorausgesetzt, können ortsfeste Server anhand des verfügbaren Datenmaterials weitere Dienste erbringen, und diese z. B. via einem Webservervice veröffentlichen. Denkbare Beispiele währen jede Art von Bildergalerien, Buddy-Systeme, u.v.a.m.

3 Plan für die Master Thesis

Die Zentralen Arbeiten meiner Master Thesis gliedern sich in drei Bereiche:

- Erweiterung des Prototypen aus dem Projekt WS05 zu einer Test-Plattform für Kooperations-Policies. Dies bedeutet vor allem auch die Erweiterung der minimalen API aus 2.2 zu einem praktisch nutzbaren System.
- Untersuchung der Fehlerformen in Kooperativen Systemen und Vergleich der unterschiedlichen Implementation, auch mit klassischen Remote-Storage Lösungen.
- Implementation von Ende-zu-Ende Verschlüsselung und Discovery in Lokalen Netzen.

3.1 Methodik

Die Implementation des Prototypen soll mit Methoden des Evolutionären Prototyping erstellt werden:

- Die Entwicklung findet Test getrieben statt, d. h. für alle Klassen und Datenstrukturen werden erst Tests, dann der eigentliche Code erstellt. Das Ziel hier ist eine Abdeckung des Codes durch Tests von mindestens 80%.
- Der gesamte Build- und Test-Prozess müssen voll automatisiert werden. Dafür kommen Werkzeuge wie Ant und JUnit zum Einsatz. Aber gerade beim Testen von verteilten Anwendungen zeigen diese Werkzeuge Schwächen oder Unzulänglichkeiten die vom Designer durchaus gewollt waren, und durch andere Werkzeuge in einer verteilten Test-Umgebung erweitert werden müssen. Diese Werkzeuge sind zu evaluieren und auszuwählen.

3.2 Risiken

Das größte Risiko für den Erfolg der Arbeit ist der Mangel an zufrieden stellenden Eigenschaften hinsichtlich Antwortzeiten, Stabilität des Gesamtsystems und der Komplexität der Handhabung sind.

Aber selbst wenn sich diese Mängel nicht beheben lassen sollten, so lassen sich dennoch Ergebnisse für weitere Arbeiten im Bereich kooperative Speicherung daraus ableiten.

Die Risiken werden versucht zu minimieren, in dem die Plattform mit Java Standard Edition gut verstanden ist, und auf die Anwendung auf den mobilen Zielgeräten verzichtet wird. Ebenso werden alle Funktionen, die nicht Bestandteil der eigentlichen Datenspeicherung und -Wiedererlangung sind, ausdrücklich ausgeklammert. Daher wird der Prototyp keine Benutzerschnittstelle haben, die in irgendeiner Form für einen End-Benutzer verwendbar ist.

Literatur

Beintner und Osius 2005 BEINTNER, Roland ; OSIUS, Hannes: *Untersuchung des Globus Toolkits als Grid Computing Middleware*. Februar 2005

Bell und LaPadula 1976 BELL, D. E. ; LAPADULA, L. J.: Secure computer systems: Unified exposition and Multics interpretation / MITRE Corporation. 1976 (588). – MTR-2997, (ESD-TR-75-306), available as NTIS AD-A023

CFS 2002 *Lustre: A Scalable, High-Performance File System*. Cluster File Systems Inc. white paper, version 1.0. November 2002. – URL `http://www.lustre.org/docs/whitepaper.pdf`. – http://www.lustre.org/docs/whitepaper.pdf

Dabek u. a. 2001 DABEK, Frank ; KAASHOEK, M. F. ; KARGER, David ; MORRIS, Robert ; STOICA, Ion: Wide-area cooperative storage with CFS. In: *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*. Chateau Lake Louise, Banff, Canada : ACM, Oktober 2001. – URL `citeseer.ist.psu.edu/dabek01widearea.html`

Ghemawat u. a. 2003 GHEMAWAT, Sanjay ; GOBIOFF, Howard ; LEUNG, Shun-Tak: The Google File System. In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. Bolton Landing, NY : ACM Press, October 2003, S. 96–108. – URL `http://www.cs.rochester.edu/sosp2003/papers/p125-ghemawat.pdf`

Goh u. a. 2003 GOH, E. ; SHACHAM, H. ; MODADUGU, N. ; BONEH, D.: *SiRiUS: Securing remote untrusted storage*. 2003. – URL `citeseer.ist.psu.edu/goh03sirius.html`

Karger u. a. 2001 KARGER, David ; BALAKRISHNAN, Hari ; STOICA, Ion ; KAASHOEK, M. F. ; MORRIS, Robert: *Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications*. November 23 2001. – URL `http://citeseer.ist.psu.edu/502954.html;http://gecko.cs.purdue.edu/gnet/papers/chord.ps`

LNCS 3779 2005 FOSTER, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems. In: *IFIP International Conference on Network and Parallel Computing*, Springer-Verlag, 2005, S. 2–13. – This paper is an excellent introduction to the Globus Toolkit 4.0 and its use.

Mazières und Shasha 2002 MAZIÈRES, David ; SHASHA, Dennis: Building secure file systems out of Byzantine storage. In: *Proceedings of the Twenty-First ACM Symposium on Principles of Distributed Computing (PODC 2002)*, URL `citeseer.ist.psu.edu/mazieres02building.html`, 2002

Oxigen 2004 OXIGEN: *Project Oxigen Network Vision*. 2004. – URL `http://www.oxygen.lcs.mit.edu/Network.html`