Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung
Anwendungen I (AI)

Lutz Behnke

Netzwerk-Transparente Persistenz

Fachbereich Informatik
Department of Computer Science

Lutz Behnke

Thema Ausarbeitung
Anwendungen I (AI)
Netzwerk-Transparente Persistenz

Stichworte
Netzwerk Persistenz Storage

Kurzzusammenfassung
Effiziente Speicherung von Daten über das Netzwerk und deren strukturierte Ablage in Datenspeichern erfordert heute eine starkes Maß an Mitarbeit des Anwenders. Dieses Papier versucht die Probleme zu beleuchten, die sich aus der Skalierung von Lösungen ergeben und stellt Vorüberlegungen zu der Frage an, ob die Komplexität der Aufgabe nicht auch für den Anwender transparent gelöst werden kann.

Lutz Behnke

Title of
Network Transparent Persistence

Keywords
Netzwork Persistence Storage

Abstract
Efficient storage of data across the network and its structured filing in data storage today requires the participation of the user to a large measure. This paper tries to shine a light on the problems that arise from the skaling of solutions and also proposes some initial ideas whether this complexity cannot be solved transparently to the user.

Inhaltsverzeichnis

1 Persistenz allgemein

Dieses Dokument stellt Überlegungen über die Ziele, Anforderungen und mögliche Technologien zur Implementation eines Netzwerk-transparenten Persistenz Systems (NTPS) an. Dabei geht es in den Anforderungen über die klassische Fähigkeiten eines Netzwerk-Dateisystems hinaus und definiert die Anforderungen an ein solches System im Kontext von Ubiqitous Computing: *anytime, anywhere, continous, unseen*

2 Zieldefinition

2.1 Definition (Netzwerk-)Persistenz

Persistenz ist von Grady Booch wie folgt in [1] definitiert:

> Persistenz ist die Eigenschaft eines Objektes durch die seine Existenz eine bestimmte Zeit überdauert (d.h. das Objekt existiert über die Existenz seines Erzeugers hinaus) und oder (Namens-)Raum durchdringt (d.h. der Ort des des Objektes entfernt sich vom Namens- oder Adressraum in dem es erschaffen wurde).

Dies erweitere ich für die Netzwerk-Persistenz um folgende Definition:

> Netzwerk-Persistenz ist erweitert diese Eigenschaft um die Fähigkeit zum richtigen Zeitpunkt die richtigen Objekte an der richtigen physikalischen Position zur Benutzung vorzuhalten, unabhängig vom physikalischen Ort der Speicherung.

2.2 Anforderungen

An ein ideales NTPS werden eine Reihe von Anforderungen gestellt, die zum Teil wiedersprüchlich sind. Ein reales System muß eine für den Anwendungszweck geeignete Annäherung an diese Ziele wählen:

Benutzer braucht Speicherort nicht kennen Im Idealfall kann der Computer ohne Hilfe des Benutzers Speicherplatz finden und Daten, die gerade nicht benötigt werden, dorthin auslagern und bei Aufruf wieder zurückholen.

Aus Gründen des Rechts, der Sicherheit und der Wirtschaftlichkeit wird der Anwender sicher aber zumindest für die Teilnahme an einer geschlossenen Gruppen authentifizieren müssen und die Gruppe vorher benennen müssen. Diese Informationen sollten aber (ausgeschlossen Authentifizierungs-Daten) einmalig auf der Anwender-Schnittstelle eingeben werden müssen und dann transparent eingesetzt werden.

Dienst ist immer und überall verfügbar Es muß die Aufgabe des Computers und nicht des Anwenders oder auch des Anwendungsprogrammierers sein, herauszufinden welcher konkreter Computer oder welche Netzwerk-Komponente zu einem gegebenen Zeitpunkt die notwendigen Dienste leistet. Als Analogie ist das Modell eines Mobil-Telefons nützlich. Ich kann andere Menschen über eine eindeutige Nummer erreichen. Wo sie sich befinden oder welche Infrastruktur ich benutzen muß um sie zu erreichen ist vollständig vor dem Anwender verborgen.

Ebenso wie in modernen Mobilfunknetzen muß auch ein Übergang von einem Netz zu einem anderen automatisch erfolgen. Eine Auswahl der verfügbaren Anbieter *kann* dem Anwender dargestellt werden. Dieser muß aber auch in der Lage sein einfach den *besten* auszuwählen und sinnvolle Kriterien müssen diesen Anbieter auswählen können.

Optimal ist es, wenn nicht nur die Client-Komponenten automatisch konfiguriert werden, sondern die Mehrheit, wenn nicht alle Komponenten Ad-Hoc zusammenarbeiten können und sich selbst Konfigurieren. Diese Fähigkeit ist vor allem bei vergänglichen Komponenten (egal ob durch technischen Defekt, Verlassen von Netzreichweiten bei mobilen Komponenten oder durch einfaches Ausschalten des PDA) notwendig.

Den Hauptunterschied zu einem Mobil-Telefon ist die gesteigerte Anforderung an die Stabilität des Systems gegen Datenverlust und gegen unberechtigten Zugriff auf die Daten. Anders als bei einem Telefongespräch, das die Partner einfach durch erneute Anwahl wieder aufnehmen können, muß das NTPS Daten, die es einmal angenommen hat, bis zur gewünschten Löschung mit hoher Sicherheit wieder zurück liefern können.

In der Definition der notwendigen Sicherheitsprotokolle stellt sich darüber hinaus die Aufgabe den Transport von Vertrauen und Berechtigungen zu gewährleisten. Dieses Vertrauen muß sowohl zwischen Menschen und Computern als auch zu Gruppen von beiden möglich sein.

Transparent für Entwickler von Anwendungen Was für den Endanwender gilt, gilt fast ebenso stark für die Entwickler von Anwendungen. Eine Technik, API oder ein Produkt wird nur dann flächendeckend eingesetzt wenn es besonders einfach zu benutzen ist. Daher muß eine Benutzung der API für den Dienst mit minimalem Aufwand möglich sein, aber auch die Mittel für eine genaue Kontrolle der Arbeit des Systems liefern.

Ebenso sollte sich die Frage, ob er gerade auf dem Server oder Client dieses Systems arbeitet, oder auch die Frage was der Unterschied zwischen Server und Client im System befindet, für den Entwickler nicht stellen, sondern einfache Konzepte und API in jedem Fall zur Verfügung stehen.

Hohe Skalierbarkeit Um den Vergleich mit einem Handy-Netz erneut aufzugreifen, ist die Skalierbarkeit des Gesamtsystems eine der überragenden Eigenschaften. Von der Architektur her sollte das System, bei einer globalen, Internet-weiten Verbreitung, in der Lage sein, Größenordnungen von 100K bis 10M Maschinen zu unterstützen. Dies verbietet nahezu automatisch den Einsatz von einzelnen zentralen Einheiten, da diese zu leicht zu überlasten und zu exponierte Ziele für Angriffe währen.

In einer *Ubiquitous Computing* Umgebung ist die Anzahl der Computer um Größenordnungen höher als die Anzahl der teilnehmenden Anwender und dennoch muß das System die Anforderungen für Daten jeder Maschine zeitnah umsetzen und die verfügbare aggregierte Netzwerk-Bandbreite effizient ausnutzen. Die Anzahl an Maschinen die heute im Internet direkt oder indirekt (etwa durch Gateways aller Art) angeschlossen sind sollten beim Design einer Architektur nur als Untergrenze angesehen werden.

3 State of the Art

Dieser Abschnitt zeigt die Systeme auf die man als Basis für ein NTPS verstehen kann, und welche Probleme sie vor der Erfüllung der in 2.2 definierten Anforderungen bieten.

Jeder dieser Technologien erfüllt eine oder mehrere der Anforderungen und können so unter Umständen zur Realisierung eines NTPS beitragen.

3.1 Netzwerk Dateisystem

Das klassische Netzwerkdateisystem ist eine weitverbreitete Implementation eines Netzwerk-Persistenz-Systems. Aber es ist eben nicht für den Anwender oder zumindest nicht für den Administrator transparent und erfordert daher vor dem Einsatz eine manuelle Vorbereitung.

Das Netz-Dateisystem ist im allgemeinen Server-zentriert. D.h. der individuelle Server (oder eine Gruppe von Servern in einem Cluster) muß vom Anwender angegeben werden. Die genaue Position für jedes Objekt in einer Objekt-Hierarchie und der Pfad dorthin muß der Anwender für jedes Objekt kennen. Natürlich kann es sich von Software dabei helfen lassen diese Namen zu merken, oder bei Bedarf zu erfahren, aber es bleibt die Aufgabe des Anwenders das gewünschte Objekt zu benennen.

In der ursprünglichen Form des Netz-Dateisystems hat jeder Server seinen eigenen Namensraum für Benutzer-Identitäten und verwaltet seine eigene Berechtigungs-Strategie. Natürlich kann diese Arbeit an andere Dienste, z.B. einen LDAP-Server delegiert werden, aber es ist die Aufgabe des Systemverwalters diese Rechte und Identitäten zu synchronisieren wenn in einer Umgebung mehrere Server eingesetzt werden. Es ist nicht Teil der Protokol-(Stack-)Definition des Netz-Dateisystems eine Server übergreifende Identität zu definieren.

Ein anderes Problem der Netz-Dateisysteme ist die Tatsache das jede Datei nur auf einem Server vorhanden ist. Bei einem Ausfall des Servers kann der Zugriff auf die Datei unter Umständen verhindert sein. Natürlich kann man dies durch

den Einsatz von redundanten Server zu verhindern suchen. Aber dies ist wiederum ein manueller Vorgang der nicht in der Lage ist Resourcen dynamisch und effizient zu verwenden. Auch Backups sind manuelle Vorgänge die zwar einen Totalverlust von Dateien unwahrscheinlicher machen können, aber nicht die Verfügbarkeit steigern. Ein NTPS sollte nicht auf Backups verzichten, sozusagen als Fall-Back Lösung, muß aber schon aus dem Design heraus die Verfügbarkeit von Daten so erhöhen das möglichst selten auf Backups zurückgegriffen werden muß.

Typischen Vertreter von Remote-Dateisystemen sind Implementation des Network File Systems (NFS) und dem Service Message Blocks (SMB), wobei letztere heute in integrierte Systeme wie dem Active Directory von Microsoft eingebunden sind, die eine Server-übergreifende Identitäts- und Rechteverwaltung liefern. Aber immer noch muß ein Systemverwalter, zum Teil sehr komplexe, Dienste aufsetzen und konfigurieren um diese Systeme zu realisieren. Und es entscheidet jeder einzelne Server auf welche Objekte er welchen Identitäten welche Operationen gestatten möchte.

Über die Frage der Konfiguration und der Rechte- und Identitätenverwaltung hinaus existiert natürlich die Frage nach der Strukturierung von Dateien und der Ablagestrategien auf der Festplatte. Dies ist aber eine Eigenschaft der Datenspeicherung auf linearen Dateisystem (flat file) und keine besondere Eigenschaft von Netz-Dateisystemen. Zu Beherschung dieses Problemes werden daher natürlich üblicherweise Datenbanken für strukturierte Daten benutzt.

3.1.1 Cluster-Dateisysteme

Klassische Netz-Dateisystem haben durch die Zentralisierung von Daten auf einem Server ein grundsätzliches Problem mit der Skalierung. Dies läßt sich in den meisten Umgebungen entschärfen wenn in dem Unter-Umgebungen mit jeweils einem lokalen Server geschaffen werden. Aber dies verletzt natürlich auch Anforderung der minimalen manuellen Konfiguration. Außerdem ist der Gewinn an aggregiertem Durchsatz nur dann akzeptabel wenn eine Untergruppe von Anwendern mit hoher Wahrscheinlichkeit auf eine Untermenge der Daten zugreift. Wenn die Wahrscheinlichkeit das ein Objekt von einem Anwender nachgefragt wird, auf alle Objekte gleichverteilt ist, so sinkt der Gewinn der lokalen Server sehr schnell, und kann durch den Overhead für eine notwendige Weiterleitung von Anfragen gegenüber einem einzelnen leistungsfähigeren Server sogar sinken.

Dieses Problem ist besonders in Systemen wie Google oder wissenschaftlichen Hoch- und Höchstleistungs-Clustern evident, da hier nicht nur Clients von 1000 und mehr Maschinen vorhanden sind, sondern auch besonders große Dateien verarbeitet werden.

Es gibt mehrere Ansätze dieses Problem zu lösen, die aber in vielen Fällen eine Eigenschaft gemeinsam haben: der Trennung von Nutzdaten und den Metadaten.

Die zugunde liegende Idee ist einfach: für die Verwaltung des Gesamtsystems sind nur die vergleichsweise sehr kleinen Metadaten notwendig, und das Schreiben und Lesen von Nutzdaten kann beim Einsatz von mehreren Speicherorten nebenläufig geschehen ohne die Integrität des Gesamtsystemes zu beeinträchtigen. Zwei beispielhafte Vertreter dieser Cluster-Dateisysteme sind Lustre [2] und das GoogleFS [3].

Mögliche Kollisionen bei Zugriffen auf Objekte werden in Netz-Dateisystemen üblicherweise mit Locks geregelt. Dabei sind diese in der Regel Objekt-global, da das Dateisystem ja keinerlei Wissen über den inneren Aufbau des Objektes hat. Theoretisch währe es natürlich möglich das Objekt als Byte-Strom zu verstehen und nur Abschnitte zu blockieren. Die Entscheidung welcher Abschnitt zu blockieren sei muß dann aber auch von der Anwendung geregelt werden.

3.1.2 Replikation

Replikation zwischen dem Server und dem Client ebenso wie zwischen mehreren Server für Fall-Back Lösungen (z.B. Coda [6]) kann helfen die Stabilität des Systemes zu steigern, da es keine permanente Netzverbindung benötigt, bringt aber eigene Probleme mit sich. Das Hauptproblem ist der Umgang mit gleichzeitigen Änderungen von Objekten während die Netzverbindung zwischen den Änwendern unterbrochen war. Coda versucht dies Problem mit Plug-In und einem Merge-Vorgang zu lösen. Aber für jedes Dateiformat müsste ein Plug-In vorhanden sein. Dies erscheint im angesicht von der Menge an Dateiformaten und vor allem undokumentierten und proprietären Dokumentenformaten wenig erfolgversprechend. Ebenso unbeantwortet ist die Frage nach der Priorität wenn in der einen Ablage ein Objekt gelöscht, in einer anderen aber noch verändert wurde.

3.1.3 Zusammegefassung

- Probleme von Remote Dateisystemen:
  - Immer Online
  - Keine Hilfe bei Objekt-Strukturen
  - Der Server ist Single-Point-Of-Failure
  - Skaliert nicht gut.
- Wege weg vom einzelnen Server:
  - Storage Area Network (SAN): Konzepte wie das Global File System (`gfs`) erlauben eine Trennung Speicherung und Betriebsystem, dann ist aber das Storage Device der SPoF. Aber es skaliert deutlich besser als einzelne Server mit dediziertem Speicher

– Server-Cluster mit speziellen Cluster-Dateisystemen wie `lustre` oder `googleFS` "Cluster" in "Cluster-Dateisystem" bezieht sich allerdings mehr auf die Anzahl der Clients, obwohl die (Meta-)Server und Speichergeräte üblicherweise auch mehrfach, verteilt und/oder geclustert ausgelegt sind. In jedem Fall schein aber eine Trennung von Daten und Metadaten die Flexibilität und Skalierbarkeit der Lösung enorm zu steigern.
– Offline: z.B. Coda um mit Replikation die Notwendigkeit einer ständigen Netzverbindung ebenso wie ein Überlastung eines einzelnen Servers zu verhindern.
– Transparenter Übergang zwischen den Server (z.B. Andrew Filesystem (AFS) [4]), erlaubt es dem Anwender zumindest den Wechsel von einem Server zum nächsten transparent durchführen zu können. Der Anwender sieht nur einen einheitlichen Namensraum für seine Objekte.

3.2 (verteilte) Datenbanken

Datenbanken liefern die Möglichkeit abgelegte Daten zu strukturieren, sind heute in den meisten Fällen Netzwerk-transparent, zumindest von mehreren Clients auf einen zentralen Server und lösen mit Erfüllung der ACID[1] Kriterien auch eine Reihe von Problemen die in 3.1 angesprochen werden.

vDBMS sind ideal geeignet wenn die Struktur der Daten bekannt ist, da sich die Konflikterkennung und -behandlung bei konkurrierenden Änderungen wie sie in 3.1.2 angesprochen werden, leichter lokalisieren und oft auch leichter lösen lassen. Das Problem von unbekannten Dateiformaten tritt hier durch die expliziete Beschreibung des Datenformates nicht auf[2].

Bei einer Verteilung der Daten auf unterschiedliche können verteilte Datenbanken auch helfen, insbesondere wenn sie, wie z.B. bei Oracle 10g, die Verteilung vollautomatisch geschehen kann. Auch die Trennung des logischen Servers von der physikalischen Maschine leisten diese vDBMS transparent.

Auch nehmen Datenbanksysteme im allgemeinen an, das ein Client von der Existenz eines bestimmten Server, für den er Zugriffberechtigungen besitz, weis. Ein automatisches Entdecken von Resourcen und Aushandeln von Diensten ist nicht üblicher Bestandteil von Datenbanksystemen. In einem vDMBS ist aber die Verbindung mit einem bestimmten (für einen normalen Benutzer nicht zu durch-

[1] Atomicity, Consistency, Isolation, Durability

[2] offensichtlich kann diese Möglichkeit durch die Ablage von Blobs mit unbekanntem Inhalt verhindert werden

schauenden[3]) Verbindungs-String an eine beliebige physikalische Instanz einer verteilten Datenbank ein Schritt in Richtung der Erfülligung der Forderungen in 2.2.

Nahezu alle Datenbanksysteme beinhalten eine eigene Benutzer- und Rechteverwaltung. Diese ist leider nur selten oder nur im begrenztem Umfang mit den Benutzerverwaltungen anderer Domänen oder Systeme zu verbinden, so das eine Definition eines Benutzers und seiner Rechte über die Grenzen von einzelnen Datenbanksystemen nicht oder nur schwer möglich ist.

Auf grundsätlicher Ebene gibt es im Moment eine Reihe von Entwicklungen bei denen Datenbank-Technologie, unter anderem der Wunsch nach Unterstützung des Atomität von Änderungen auch in Dateisystemen umzusetzen. Ein interessanter Ansatz dazu findet sich in `ReiseFS` und seinem Nachfolger `Reiser4`[5]

3.3 RMI/ORB-Persistenz

Mit allen Dienst-orientierten Komponenten-Technologien, egal ob RMI, DCOM+, CORBA oder auch Web-Services lassen sich entfernte Persistenz realisieren, bei denen der Benutzer dieser Dienste nicht wissen muß ob der Persistenz-Dienst entfernt ist oder nicht.

Und in CORBA und Web-Services sind auch Protokolle und Repositories definiert mit denen ein Prozess während der Laufzeit einen Persistenz-Dienst finden und kontaktieren kann. Aber hier stellt sich die Frage was ein Persistenz-Dienst ist und welche Eigenschaften er hatt. In globalen Diensteverzeichnissen wie dem UDDI stehen viele Dienste, aber welcher davon ist ein Persistenz-Dienst?

Für CORBA hat die OMG diese Frage eigentlich beantwortet, da es nicht nur eine genaue Beschreibung der Schnittstellen, sondern sogar eine spezielle Untersprache zur Interface Definiton Language (IDL) gibt, welche die Eigenschaften eines Persistenz-Dienstes beschreibt. Diese Beschreibungssprache ist aber leider so allgemeingültig, das sich die Dienste die damit beschreibbar sind, nur schwer automatisch vergleichen lassen. Dieses währe aber notwendig, um transparent Komponenten und Dienstanbieter vom Computer auswählen zu lassen.

Ohnehin wird CORBA für Dienste, die über Organisationsgrenzen hinweg Verbindungen aufbauen, nur selten eingesetzt. Dies hat aber mehr historische, organisatorische und Produkt-/Anbieter- denn technische Gründe. Es währe alles mit den vorhandenen Spezifikationen machbar. Es tut aber keiner.

Webservices (WS) reimplementieren den Hauptteil der in CORBA bereits vorhandenen Protokolle (auf eine weniger effiziente Art und Weise) aber propagieren von Anfang an das Ziel über Organisationsgrenzen hinweg zu operieren. Aber es

[3] Auch URLs sind für die meisten normalen Benutzer lange, unstrukturierte Zeichenketten, die *vielleicht* mnemonische Hilfsmittel wie “media-markt” oder “amazon.de” enthalten.

exisitert noch keine Spezifikation für einen WS-Persistenz-Dienst.

3.4 Peer2Peer Netze

Peer-2-Peer Netze ohne zentralen Server wie eDonkey, Gnutella[4] , BitTorrent[5] oder Mute[6] liefern eine Reihe von wünschenswerten Eigentschaft für ein perfektes transparentes Persistenz-Systems:

- Keine feste Bindung zwischen Objekt und Speicherort/physikalischem Server mehr. Es können sowohl mehrere Server zur Übertragung von Daten beitragen als sich auch der Speicherort während der Übertragung an den Client ändern kann.
- Es existiert keine feste, hierarchische Ordnung zwischen (wenigen) Servern, evtl. Proxies und (vielen) Clients. Alle Teilnehmer sind in erster Näherung gleichberechtigung. Nur die Leistungsfähigkeit macht einige gleicher als andere.
- Nodes können das System jederzeit betreten oder verlassen ohne das dies die Funktionalität des Gesamtsystemeses beeinträchtigen würde oder auch nur manuelle Konfigurationen notwendig währen.

Dies wird vor allem durch eine verteilte Datenbank erreicht in der alle Objekte abgelegt werden, nach denen dann gesucht werden kann. Es existiert keine heirarchische Speicherung von Objekten wie in einem Dateisystem. Und da kein Knoten alle Informationen haben kann, hängt die Qualität eines P2P Systems vor allem von der Qualität der verteilten Suche ab die auf jedem Knoten nach Objekten mit den gewünschten Eigenschaften sucht.

Die oben genannten Eigenschaften werden aber auch mit einer Reihe von Nachteilen erkauft:

- Der Prozess mit dem ein Objekt in das System eingebracht wird ist nicht Bestandteil des P2P Systems sonder wird auf jedem Knoten lokal gesteuert. Das System ist aus Sicht des Clients read-only.
- Ebenso wie das Anlegen eines Objektes ist auch das Löschen eines Objektes ein Prozess unter Knotenkontrolle und kann vom Client nicht verhindert werden. Will der Client sicherstellen das ein Objekt zu einem späteren Zeitpunkt noch verfügbar ist muß er es herunterladen und lokal speichern.

[4] http://www.gnutella.com/

[5] http://www.bittorrent.com/introduction.html

[6] http://mute-net.sourceforge.net/

Forschung in diesem Bereich liefert gute Beispiele für largest-Scale verteilte Datenbank-Technik mit distributed-Hash Algorithmen. Chord[7] [7] liefert ein Beispiel für eine Basis-Technologie als Träger für weitere Entwicklungen.

3.5 Suche auf Metadaten

Offensichtlich habe viele Menschen den Überblick über ihre Dateien verloren. Denn eine gante Reihe von namhaften Anbietern konzentrieren sich derzeit darauf, Objekte nicht nach ihrer Einordnung in eine Verzeichnis-Hierarchie zu identifizieren, sondern nach Metadaten. Dabei lassen sich neben generische Metadaten wie Erstellungs-, Änderungsdatum, Namen, Objekt-Typ oder Ersteller auch objektspezifische Eigenschaften heranziehen. Z.B das Format (Höhe mal Breite) eines Bildes oder die durchschnittliche Bewertung eines Musikstückes durch meine Freunde[8].

4 Vorschlag für Projekt

Mein Vorschlag für das Projekt: Ich stelle eine Schnittstelle bereit mit der sich Objekte ablegen lassen, bei Bedarf wieder laden lassen und mit Hilfe von Metadaten suchen lassen. Dabei werden so Details wie Speicherort oder ähnliches vom Computer gehandhabt. Auch die Frage ob ein lokaler Speicher langsam überläuft und bei nächster Gelegenheit alle Objekte auf die schon lange nicht mehr zugegriffen wurde in einen anderen Datenspeicher verschoben werden sollten, müssen nicht vom Benutzer beantwortet werden.

Natürlich muß das ganze mit einem Berechtigungs- und Sicherheitsmodell versehen werdem um sicher zustellen das keine Objekte verschwinden oder unberechtigte Zugriff nehmen können.

4.1 Motivation

Die Erfüllung der Anforderungen in 2.2 ist eigentlich Motivation genug. Ein Benutzer soll sich nie wieder Sorgen um Speicherplatz machen. Festplatten sind billig genug. Die Aufgabe Platz zu finden kann man viel besser dem Computer überlassen.

4.2 Schnittstelle

Eine aller erste und völlig unzureichende Version der Schnittstelle:

[7] http://pdos.csail.mit.edu/chord/

[8] wobei dann natürlich erst einmal geklärt werden muß wer meine Freunde sind

```
interface Persistence {
    struct genericObject {
       long    owner;
       long    time;
       octet[] data;
     };

    typedef octet[160] OpaqueKey;

    exception InvalidKeyException{};

    OpaqueKey registerObj(in genericObject obj);
    void retrieveObj(in OpaqueKey key, genericObject obj)
            raises InvalidKeyException;
};

interface AbstractGenerator {
        void retrieveAbstract(in OpaqueKey key)
                raises  InvalidKeyException,
                        ObjTypeUnkonwnException;
};
```

5 Ideen

Dies sind ein paar Ideen und Erkenntnise die mir im Zusammenhang mit der Bearbeitung des Themas gekommen sind.

5.1 Möglichkeiten wenn Speicherort und Name von Objekten vollständig getrennt sind.

Wenn der Speicherort und -art eines Objektes vollständig getrennt sind ergeben sich die Möglichkeit einige Operationen zu delegieren, vorrausgesetzt die Aufrufende Applikation delegiert ihrerseits diese Aufgaben.

- Das kopieren von Objekten von einer Maschine auf eine andere. Wenn die Anwendung auf Maschine A läuft und Daten von B nach C kopieren will. Wenn die Applikation nicht versucht dies selbst durchzuführen, sondern die Funktion dem Betriebsystem oder der Middleware überläßt.

  In einem perfekten System währe aber auch der Kopiervorgang nicht mehr notwendig, da alle Beteiligten nur noch Identifikatoren für die Datei austau-

schen müssten und sie bei Bedarf vom nächstgelegenen Speicherort laden würden.

- Delegation von Dateimanipulationen auf eine andere Maschine. Da eine Applikation ohnehin nich feststellen kann wo ein Objekt abgelegt ist, lohnt es sich Änderungsoperationen zu definieren und Änderungen oder Änderungsanweisungen in Richtung des Objektes zu schicken, anstatt das Objekt für die Änderung zur lokalen Maschine zu holen.

- Redundanz muß anhand von Heuristiken und Service Level Agreements durchgeführt werden. Wer ein Objekt in seinem Speicher hält ist dafür verantwortlich und sollte sich eine Reihe von anderen Speicherorten suchen die im Notfall als Alternativen angegeben werden können. Dies ist eine notwendige Erweiterung von klassischen Trust-Beziehungen in verteilten Systemen.

5.2 Änderungen eines Objektes

Ich halte es für beinahe unmöglich, in einem lose gekoppelten System, ein globales Locking zu erreichen. Ein derartiges System scheint mir aber als einziger geeigneter Lösungsansatz um die unter 2.2 genannten Anforderungen zu erfüllen. Damit können Änderungen an Objekten nicht kontrolliert werden.

Mir erschein die Definition das Objekte niemals geändert werden, sondern höchsten in ein neues Objekt transformiert werden als einzige sinnvolle Lösung.

Ein Verbindung von einer Änderung zur nächsten kann besser als Meta-Information abgebildet werden. Da eine leistungshähige Verwaltung von Metadaten ohnehin ein notwendiges Merkmal zu seien scheint (siehe auch die Ankündigung von Microsoft für Longhorn eine SQL-Datenbank als Dateisystem einsetzen zu wollen.). Die Änderungen können so als Versionierung modelliert werden.

5.3 Abstrakt-Generatoren

Gerade bei Mediendaten, wie Filmen oder Musik oder auch bei Dokumenten, läßt sich oft eine Menge sinnvolle Mediendaten direkt aus dem Objekt generieren wenn man dessen inneren Aufbau kennt. Da dies aber von jeder generierenden Anwendung abhängt, erscheint es wenig sinnvol diese Information in das NTPS zu integrieren. Daher gibt es eine Schnittstelle die andere Anwendungen und Komponenten dem NTPS zur Verfügung stellen können: den Abstrakt-Generator. In diese Schnittstelle wird die Referenz auf ein Objekt gegeben und ein (objekt spezifischer) Satz von Meta-Daten zurückgegeben.

Mit dieser Methode können beliebige Metadaten generiert werden. Und wenn diese Schnittstelle aufgerufen wird wenn ein Objekt an das NTPS übergeben wird, ist es wahrscheinlich daß das Objekt noch für den Erzeuger lokal geschspeichert ist. Wenn der Erzeuger auch die Aufgabe des Abstrakt-Generators übernimmt kann so der Vorgang sehr effizient gestalltet werden. Sollte diese Annahme nicht zutreffen so kann sich der Abstrakt-Generator wie jeder andere Client auch das Objekt durch das NTPS zur Verfügung stellen lassen.

5.4 Berechtigungen

- Standardfall
  - Alle Objekte sind verschlüsselt.
  - Alice legt ein Objekt ab.
  - Alice gibt Bob den Schlüssel
  - Bob lädt das Objekt und entschlüsselt es.

  Das Objekt muß als für die Gruppe (Alice,Bob) als kompromitiert gelten. Ein Entziehen des Schlüssel ist wenig sinnvoll, da davon auszugehen ist das beide das Dokument im Klartext besitzen.

Literatur

[1] Grady Booch. *Objektorientierte Analyse und Design*. Addison-Wesley, 1995.

[2] Inc. Cluster File Systems. Lustre: A scalable, high-performance file system, November 2002.

[3] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 96–108, Bolton Landing, NY, October 2003. ACM Press.

[4] John H. Howard. On overview of the Andrew File System. In USENIX Association, editor, *USENIX Conference Proceedings (Dallas, TX, USA)*, pages 23–26, Berkeley, CA, USA, Winter 1988. USENIX.

[5] Hans Reiser. Reasons why reiser4 is great for you, 2004. Dies ist kaum ein wissentschaftlicher Aufsatz, sondern ziemliche Selbst-Beweiräucherung. Aber es beschreibt eine Reihe von Eigenschaften die Reiser4 gegenüber klassischen Dateisystemen besitzt.

[6] M. Satyanarayanan. Coda: A highly available file system for a distributed workstation environment. In *Second IEEE Workshop on Workstation Operating Systems*, Pacific Grove, CA, USA, September 1989. http://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html.

[7] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.