OceanStore

Johann Latocha

12. März 2007

Inhaltsverzeichnis

Zusammenfassung

OceanStore[1] ist ein Konzept für ein globales Speichersystem dem Sicherheit und Verfügbarkeit als zentrale Designmerkmale zu Grunde liegen. Die Verwendung eines Mediums, das auf unsicheren Servern aufbaut, hat bestimmte Anforderungen, die durch Kryptographie und Redundanz erfüllt werden. Da OceanStore eine Peer-to-Peer Architektur zu Grunde liegt, gilt es sowohl das Problem der Performanz als auch der Lokalisierung der Daten zu lösen. Ein bereits existierender Prototyp wurde in der Programmiersprache Java entwickelt.

1 Überblick und Motivation

Rechner sind in der heutigen Zeit nicht nur auf Schreibtischen in Büros oder Arbeitszimmern zu finden. Der Zugriff auf bestimmte Daten erfolgt zunehmend auch von mobilen Geräten wie Handys, PDAs und natürlich auch Notebooks, so dass die Frage nach einer geeigneten Methode zur Speicherung aufkommt, die einen komfortablen Abruf der Daten ermöglicht.
In diesem Artikel soll das globale Dateisystem OceanStore vorgestellt werden, das von einem seiner Autoren folgendermaßen beschrieben wird: ,,servers in the world form an 'ocean' of data. This data quickly 'flows' to where it is needed"[3]. OceanStore wird an der University of California, Berkeley als Projekt entwickelt und es exisitiert bereits ein Prototyp, dessen Quellcode frei verfügbar ist[2].

1.1 Dateisysteme

Dateisysteme lassen sich im Allgemeinen in *lokale* (zB. FAT,ext2) und *verteilte* klassifizieren, wobei sich die verteilten Systeme wieder in drei weitere Kategorien unterteilen lassen: *Remote Dateisysteme*, *Cluster Dateisysteme* und *Globale Dateisysteme*.

1. *Remote Dateisysteme*, auch Netzwerk-Dateisysteme genannt, werden zentral durch einen Server als Dienst bereitgestellt. Das wohl bekannteste Beispiel hierfür ist die von Microsoft eingeführte ,,Datei- und Druckerfreigabe" (auch samba), jedoch können FTP[3] und NFS[4] auch dazu gezählt werden.

2. Wenn sich ein Remote Dateisystem über mehrere Rechner eines lokalen Netzwerks erstreckt, wird es *Cluster Dateisystem* genannt. Wichtig hierbei ist, dass alle teilnehmenden Server als vertrauensvoll eingestuft werden müssen.

3. *Globale Dateisysteme*, zu denen auch OceanStore zählt, reichen über das lokale Netz hinaus. Als Medium wird das Internet eingesetzt, das vielerlei Vorteile aber auch Probleme mit sich bringt auf die wir in den nächsten Abschnitten eingehen werden.

[1] http://oceanstore.cs.berkeley.edu
[2] http://oceanstore.sourceforge.net/
[3] File Transfer Protocol, http://tools.ietf.org/html/rfc959
[4] Network File System, http://tools.ietf.org/html/rfc3530

1.2 Vorteile globaler Dateisysteme

Ein enormer Vorteil globaler Dateisysteme ist die hohe Speicherkapazität, die durch die fast unzähligen Rechner im weltweiten Datennetz bereitgestellt werden kann. Praktisch jeder Personal Computer, egal ob Server in einem Rechenzentrum oder ein einfacher Desktoprechner eines Privatanwenders, kann ein Teil des Dateisystems werden. Aus diesem Szenario wird direkt ein weiterer Vorteil ersichtlich: Eine hohe Verfügbarkeit ist durch die dezentrale Architektur gegeben, da somit Angriffe (zB. DOS[5]) kein direktes Ziel haben. Berücksichtigen sollte man auch den Aspekt, dass Daten trotz einer Beschädigung oder gar Zerstörung des konsumierenden Endgerätes auf Grund dauerhafter Archivierung nicht verloren gehen. Ein weiterer Punkt, der durch den zunehmenden Einsatz mobiler Endgeräte immer wichtiger wird, ist die Möglichkeit des Zugriffs auf benötigte Daten unabhängig vom aktuellen Standort des Benutzers. OceanStore ist so konzipiert, dass dies unter der Voraussetzung eines vorhandenen Internetanschlusses von jeder Stelle des Netzwerks aus möglich ist. Es ist auch vorstellbar, dass der Betrieb des Systems durch verschiedene Organisationen erfolgen kann. So könnte ein Webspace-Anbieter Speicherplatz zur Verügung stellen, den er selbst von anderen Organisationen bezieht.

1.3 Einsatzmöglichkeiten

OceanStore ist laut Schätzung der Autoren in der Lage 10^{10} Benutzer mit jeweils 10000 Dateien zu verwalten[2]. Das macht letztlich ein Datenvolumen von 10^{14} Dateien aus!
Eine Anwendung findet man in der Archivierung großer Datenbestände, die zum Beispiel von digitalen Bibliotheken vorgenommen wird. Aber auch die Art und Weise, wie auf die Daten zugegriffen wird, spielt eine Rolle. So ist es für Groupware-Dienste von Vorteil, wenn ein Zugriff unabhängig vom Standort erfolgen kann. Zu diesen Diensten zählen verteilte Kalender und Adressbücher, aber auch dem gemeinsamen Projektmanagement sind somit keine „Grenzen“ gesetzt.

1.4 Probleme

Eines der schwerwiegendsten Probleme globaler Dateisysteme ist die Gefahr des Ausfalls eines teilnehmenden Rechners. Durch die starke Fragmentierung des Datenbestandes ist Datenverlust eine ständige Bedrohung, die beseitigt werden muß.
Auch der Datendiebstahl ist durch die nahezu anonyme Identität und dem leichten Zugang zum Internet ein bekanntes Problem, vor dem Benutzer verteilter Dateisysteme geschützt werden müssen.
Durch die immense Vielfalt vorhandener Rechner tritt ein neues Problem auf: PCs, die an das weltweite Datennetz angeschlossen sind, unterscheiden sich sowohl stark in ihrer Leistungsfähigkeit als auch in der Bandbreite der Netzwerkanbindung. Somit muss auch mit einer möglichen Ineffizienz gerechnet werden, die die Arbeit auf einem global verfügbaren Dateisystem im Alltag stark behindert.

[5] Denial of Service

2 Designmerkmale

Das Kernkonzept von OceanStore umfasst bei stark abstrakter Sichtweise zwei zentrale Designziele: *Sicherheit* und *Verfügbarkeit*. Ersteres soll das Problem möglicher Ausfälle umgehen und dem Datendiebstahl entgegenwirken. Das zweite Ziel soll sowohl eine Lösung für das Ineffizienzsproblem bieten, auch der Ausfallgefahr entgegenwirken.

2.1 Sicherheit

Wie bereits in der Motivation erwähnt ist das Ziel global verfügbarer Dateisysteme trotz einer unsicheren Umgebung das korrekte Verhalten des Systems zu garantieren. Dies wird zum einen durch die Verwendung von *Kryptographie*, dem Gebrauch von *floating replicas* sowie durch das Konzept der *Deep Archival Storage* erreicht.

1. Daten werden in OceanStore mit Hilfe von symmetrischen Verschlüsselungsverfahren codiert. Damit lässt sich eine Zugriffsbeschränkung realisieren, um private Inhalte zu schützen. Kryptographische Hash-Funktionen (SHA-1[6]) werden zur eindeutigen Identifikation verwendet, die ein Wiederfinden der eigenen Datei ermöglichen.

2. Um einen möglichen Datenverlust vorzubeugen setzt man in OceanStore auf eine Redundanz des Inhalts. Mit *floating replicas* geht man einen Schritt weiter, indem man Replikate vorhandener Daten nicht an einen physikalisch gegebenen Server bindet.

3. Das Modell der *Deep Archival Storage* ermöglicht eine Versionskontrolle, die bei eingetretenen Fehlern eine ,,undo"-Funktionalität bereitstellt. Auch dies trägt zur Sicherheit bei.

2.2 Verfügbarkeit

Der zentrale Begriff dieses Abschnitts ist *nomadic data*. Dies meint, dass die Serverunabhängigkeit dadurch realisiert wird, dass gespeicherte Daten immer in Bewegung gehalten werden. Diese Idee soll es ermöglichen, dass Daten immer genau dort sind, wo sie gebraucht werden[3]. Bei näherer Betrachtung dieser Ausgangssituation fällt das Problem der Ineffizeinz besonders stark ins Gewicht, da die häufige Bewegung oft Engpässen des Netzwerks begegnen muss. Diesem will man mit *promiscuous caching* gezielt entgegen wirken, das die Möglichkeit bietet, Daten überall und zu jeder Zeit cachen zu können. Die Optimierung dessen ist durch das *introspective monitoring* spezifiziert, das jedoch in der aktuellen Implementierung von OceanStore noch nicht integriert ist.

3 Datenmodell

In diesem Abschnitt behandeln wir das Datenmodell, also die Repräsentation und den Aufbau der Daten in OceanStore. Das Modell bietet einige interessante

[6] Secure Hash Algorithm 1, http://tools.ietf.org/html/rfc3174

Funktionen wie die Versionskontrolle und unterschiedliche Zugriffskontrollmechanismen für Lese- und Schreiboperationen. Aber auch triviale Merkmale wie das Ändern bestehender Daten, das nicht selbstverständlich ist da z.B. das globale Dateisystem PAST dazu nicht fähig ist[6], werden in diesem Abschnitt betrachtet.

3.1 Aufbau

Daten werden in OceanStore durch sogenannte Datenobjekte repräsentiert, die außer des zu speichernden Inhalts eine Menge benötigter Informationen beinhalten. Die eindeutige Identifikation erfolgt über eine GUID[7], die ein Hashwert (SHA-1) über den öffentlichen Schlüssel des Besitzers, dem Namen der Datei und dem eigentlichen Inhalt ist. Bei der Wahl von 160 Bit für die GUIDs wird die Wahrscheinlichkeit eines Namenskonflikts nach dem Birthday-Paradoxon erst ab einer Anzahl von 2^{80} Objekten relevant.
Datenobjekte werden fragmentiert und über die teilnehmenden Rechner verstreut, wobei der Grad der Fragmentierung variiert. Wenn eine Anfrage an einen Computer nach einer bestimmten GUID gerichtet wird, erhält man eine Referenz auf ein sogenanntes Wurzelobjekt, das weitere Referenzen enthält. Dieses Modell weist starke Ähnlichkeit zum bekannten Inode-Konzept, das in Dateisystemen vieler Unix-Betriebssysteme eingesetzt wird (zB. ext2/3). Die Daten selbst werden in Blöcken organisiert, wobei der Zugriff auf diese durch B-Bäume erfolgt und in logarithmischer Zeit realisierbar ist. Wurzelblöcke enthalten zusätzlich noch Metainformationen M, das können Baumparameter, Besitzer und ein MIME-Typ sein.

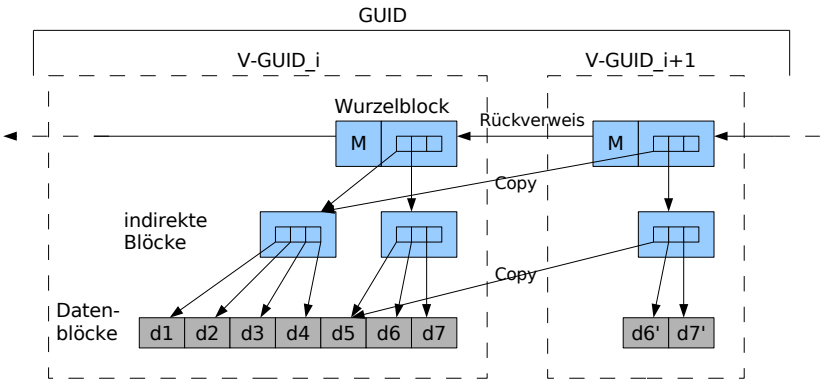

Abbildung 1: Aktives Datenobjekt nach [4]

Der Schreibvorgang auf eine Datei wird stets nur auf den geänderten Datenblöcken durchgeführt, wobei immer eine neue GUID entsteht und die alte als V-GUID[8] gespeichert wird. Ein neuer Wurzelknoten entsteht, der neu angelegte sowie die unveränderten alten Blöckt referenziert. Siehe Abbildung 1.

[7] Globally Unique Identifier
[8] Version-GUID

3.2 Der innere Ring

Der innere Ring setzt sich aus einem Verbund von leistungsfähigen Servern zusammen, dessen Hauptaufgabe darin besteht, die aktuelle Version eines Datenobjektes bereit zu stellen und Änderungen an diesen zu speichern. Wenn ein Benutzer nach einer Datei fragt, wird zuerst der lokale Cache durchsucht. Bei Misserfolg wird diese aus dem inneren Ring heruntergeladen, wobei die Möglichkeit gegeben ist, stets eine serialisierte (dh. defragmentierte) Form einer Datei aufrecht zu erhalten. Folgende Bedingungen sollten vom inneren Ring erfüllt sein:

- Der innere Ring sollte aus vertrauenswürdigen Servern bestehen, da diese zum Einen immer die aktuelle Kopie des Datenobjektes bereitstellen und zum Anderen entscheiden, wer Schreibzugriff hat und wer nicht (s. Abschnitt 3.4).
- Andererseits müssen die Computer viel Rechenleistung besitzen, da das Serialisieren der Datenobjekte eine anspruchsvolle Aufgabe darstellt.
- Eine gute Anbindung an das Internet und eine leistungsfähige Vernetzung der Rechner untereinander ist von Vorteil, da der innere Ring von vielen Benutzern gleichzeitig beansprucht wird.

3.3 Der äußere Ring

Im Gegensatz zum inneren werden an den äußeren Ring keine Anforderungen gestellt. Zu diesem können sämtliche im Internet verfügbaren Rechner gehören, der PC einer Privatperson ebenso wie ein Server im Rechenzentrum. Die Aufgabe dieser an OceanStore teilnehmenden Rechner besteht in der Speicherung von Daten, genauer der Archive. Da auf alten Versionen einer Datei nicht geschrieben werden kann, müssen die Daten auch nicht serialisert werden. Daraus folgt, dass auch leistungsschwache Rechner zulässig sind. Im äußeren Ring herrscht ein hoher Grad an Fragmentierung, jedoch strebt das System stets nach einer gleichmäßigen und damit fairen Verteilung der einzelnen Fragmente auf die verschiedenen Hosts.

3.4 Zugriffskontrolle

OceanStore besitzt eine eigene Rechteverwaltung, die durch den starken Einsatz von Kryptographie geprägt ist. Grundsätzlich wird zwischen Lese- und Schreibzugriffen unterschieden, wobei die Realisierung einer Lesebeschränkung keine große Herausforderung darstellt, die Kontrolle der Schreibzugriffe dafür umso mehr.

3.4.1 Lesezugriff

Alle nicht öffentlichen Daten werden in OceanStore verschlüsselt (symmetrisches Verschlüsselungsverfahren). Damit wird eine simple Lesebeschränkung geschaffen, denn nur wer den Schlüssel besitzt, kann die Daten entschlüsseln und somit lesen. Will man nun einem Bekannten das Lesen eigener Inhalte erlauben, muss man ihm den Schlüssel zur Verfügung stellen. Da OceanStore für jedes Objekt einen anderen Schlüssel zuläßt, ist die Erzeugung einer Gruppenstruktur

möglich, indem man einen Schlüssel für öffentliche, einen für private Daten usw. verwendet. Soll einem Benutzer der Lesezugriff wieder entzogen werden, so muss dies durch das erneute Kodieren der Daten mit einem neuen Schlüssel erfolgen. Da man in OceanStore das Löschen der Archive nicht erzwingen kann, bleiben diese für Benutzer, die im Besitz des alten Schlüssels sind, weiterhin lesbar, was aber kein OceanStore spezifisches Problem ist. Somit läßt sich zusammenfassend sagen, dass die Kontrolle der Lesebeschränkung bei den Clients liegt.

3.4.2 Schreibzugriff

Die Kontrolle über Schreibzugriffe ist hingegen serverseitig. Der Besitzer einer Datei kann dem Server (bzw. dem Serververbund, da Schreiboperationen nur auf dem inneren Ring zulässig sind) eine signierte Access Control List (ACL) senden, die eine Auflistung aller berechtigten Teilnehmer enthält. Bei einem Schreibzugriff wird eine mit einem Zertifikat signierte Kopie an den inneren Ring geschickt. Diese Server vergleichen die angehängte Signatur mit der ACL, wobei bei einer Übereinstimmung das Update erfolgt und andernfalls einfach verworfen wird.

4 Updates und Archivierung

Dieser Abschnitt behandelt das Updatemodell sowie die Archivierung in Einem, da diese beiden Themen in OceanStore eng miteinander verbunden sind. Die Archivierung wird mit Hilfe einer Sonderkodierung im äußeren Ring vorgenommen. Für das Updaten sind generell folgende Operationen erlaubt:

- Das Löschen vorhandener Blöcke
- Das Ersetzen von Datenblöcken
- Die Möglichkeit, neue Daten anzuhängen

Diese Operationen werden ausschließlich auf verschlüsselten Daten ausgeführt, so dass ein serverseitiges Mitlesen nicht möglich ist.
Tritt der Fall ein, dass mehrere konkurrierende Updateanfragen anstehen, so wird ein Replikat im inneren Ring serialisiert, um danach alle Transaktionen atomar auszuführen (Einhaltung der ACID[9]-Eigenschaften). Um auszuschließen, dass ein Server die komplette Kontrolle über die Daten erhält, sind am Serialisierungsvorgang mehrere Server beteiligt. Da OceanStore aber auf einer nicht vertrauenswürdigen Infrastruktur aufbaut, kommunizieren die beteiligten Server im inneren Ring nach dem *Byzantinischen Vereinbarungsprotokoll*.

4.1 Byzantinisches Vereinbarungsprotokoll

Byzantinische Fehler sind solche, bei denen Teilnehmer (absichtlich) verfälschte Nachrichten schicken. Der Name geht auf das Problem der Byzantinischen Generäle[10] zurück, für das der folgende Satz gilt:

Satz 1 *Das Byzantinische Generäle Problem ist nur dann lösbar, wenn weniger als ein Drittel der beteiligten Prozesse fehlerhaft ist.*

[9] Atomicity, Consistency, Isolation and Durability
[10] http://research.microsoft.com/users/lamport/pubs/byz.pdf

Anders ausgedrückt bedeutet dies, dass bei n Servern für $n = 3m + 1$ mehr als m Server „böse“ Absichten haben müssen, um das *Byzantinische Vereinbarungsprotokoll* zum Scheitern zu bringen.
Dieses Protokoll ist leider sehr kommunikationsintensiv (Bei n Rechnern sind n^2 Nachrichten notwendig), weil jeder mit jedem „reden“ muß. Deshalb wird in OceanStore eine abgewandelte Form verwendet, die mit Hilfe von Kryptographie arbeitet (Castro-Liskov-Algorithmus)[4]. Hierbei werden die Nachrichten vom Absender signiert, so dass der Kommunikationsaufwand sinkt.

4.2 Deep Archival Storage

Wie am Anfang erwähnt, sind die Archivierung und das Aktualisieren der Daten eng miteinander verbunden. Das liegt daran, dass jeder einzelne Updatevorgang, wie im Abschnitt 3.1 gezeigt, eine neue Version des Dokuments generiert und die alte in den äußeren Ring verschiebt.
Ein simples Vorgehen, um die Verfügbarkeit archivierter Daten zu steigern, ist das Herstellen von Redundanz. Bei einer 1:1 Kopie ist allerdings höchstens ein Fehler erkennbar wobei der doppelte Speicherplatz verbraucht wird. Um die Fehlertoleranz der Langzeitarchivierung in OceanStore zu optimieren, werden so genante *Erasure Codes* eingesetzt. Diese haben im Vergleich zu simplen Replikaten bei gleichem Speicherverbrauch eine höhere Fehlertoleranz. Hierbei wird ein Datenblock in m Fragmente unterteilt, die wiederum in n Fragmente kodiert werden. Dabei gilt $n > m$. Bei einer Kodierungsrate von $r = \frac{m}{n}$ erhöht sich der Speicherbedarf um den Faktor $\frac{1}{r}$, wobei der ursprüngliche Datenblock aus m beliebigen Fragmenten rekonstruiert werden kann.
Da das Berechnen von *Erasure Codes* im Normalfall sehr rechenaufwändig ist, wird im Prototyp „Pond“ der *Cauchy Reed Solomon Code* eingesetzt[4]. Dieser überführt die nötigen Berechnungen in XOR-Operationen[1].
Als Beispiel sollen uns die Zahlen dienen, die tatsächlich in „Pond“ Verwendung finden [4]: Ein Datenblock wird in 16 Fragmente unterteilt ($m = 16$). Diese werden wiederum in 32 Fragmente kodiert ($n = 32$) was eine Kodierungsrate von $\frac{1}{2}$ ergibt. Für die Rekonstruktion werden nur 16 Fragmente benötigt, nun aber auf wesentlich mehr Arten erzeugt werden können. Beim Betrachten der Anzahl möglicher Permutationen wird dies deutlich.

5 Routing

OceanStore baut auf einer Peer-to-Peer Architektur auf, so dass es keine zentrale Steuereinheit gibt, die alle Knoten des Netzwerks kennt. Dies führt zu der fundamentalen Frage, wie ein Datenobjekt zuverlässig gefunden werden kann, das auf einem belibigen Knoten des Netzwerks gespeichert ist, und auch wie Anfragen an dieses weiterzuleiten sind. OceanStore überführt hierbei das Lokalisieren und Routen in eine Operation.
Ein guter Routingmechanismus sollte den folgenden Bedingungen genügen:

- *Deterministisches Auffinden:* Wenn ein Objekt im Netz existiert, muss es auch gefunden werden. Die Erfüllung dieser Forderung ist nicht selbstverständlich (zB. in Gnutella[11] und Freenet[12] nicht realisiert).
- *Lokalität des Routings:* Die durchlaufenen Wege sollten möglichst minimal sein.
- *Minimalität und Lastverteilung:* Die Lastenverteilung auf teilnehmende Knoten sollte gerecht sein. Unnötige Last ist zu vermeiden!
- *Dynamische Mitgliedschaft:* Die Anzahl der teilnehmenden Knoten ist stets variabel, dh. Knoten können zu jeder Zeit das Netz betreten und wieder verlassen (zB. von Plaxton nicht erfüllt, siehe 5.2).

5.1 Zwei Strategien

OceanStore setzt beim Routing auf zwei verschiedene Strategien. Der *lokal-stochastische Ansatz* wird stets zuerst versucht und nur wenn dieser scheitert, greift der *global-deterministische Ansatz*, der das gesuchte Objekt auf jeden Fall findet. Die Laufzeit dieser Kombination ist nicht sonderlich größer als die, die der deterministische Algorithmus im Alleingang in Anspruch nimmt, auch wenn der stochastische Ansatz scheitert!

1. *Der lokal- stochastische Ansatz:* Mit lokal ist hier die nähere Umgebung des Knotens gemeint, der eine Suche startet. Dieses Verfahren wird eingesetzt, um die Geschwindigkeit beim Routing zu steigern, da deterministische Verfahren bei nahe liegenden Knoten eine schlechtere Performanz aufweisen. Jedoch wird hierbei nicht jedes existierende Objekt gefunden, so dass nach erfolgloser Suche mit dem deterministischen Ansatz fortgefahren werden muss.
2. *Der global- deterministische Ansatz:* Der deterministische Algorithmus zur Lokalisierung gesuchter Objekte arbeitet global, also auf dem gesamten Netzwerk. Hierbei ist ein strukturiertes Vorgehen nötig, um die zu Beginn des Abschnitts 5 vorgestellten Bedingungen zu erfüllen, insbesondere den des *deterministischen Auffindens*.

[11] http://rfc-gnutella.sourceforge.net/
[12] http://freenetproject.org/

5.2 Plaxton als Grundlage

Das Plaxton Schema dient, trotz der fehlenden Unterstützung der *Dynamischen Mitgliedschaft*, als Grundlage für das im nächsten Unterabschnitt vorgestellte *Tapestry*.
Jeder Knoten bekommt eine *node-ID* fester Länge, die trotz gleichem Namensraums nicht mit der GUID verwechselt werden darf. Bei der Suche nach dem richtigen Weg entscheidet immer die größtmögliche ID-Übereinstimmung, wobei die *Neighbor-Map* jedes Knotens in Level unterteilt ist. Level 1 bedeutet, in unserem Beispiel von rechts gelesen, dass keine Übereinstimmung gegeben ist. Ab Level 2 ist die letzte Stelle gleich und ab Level 3 bereits die letzten zwei.

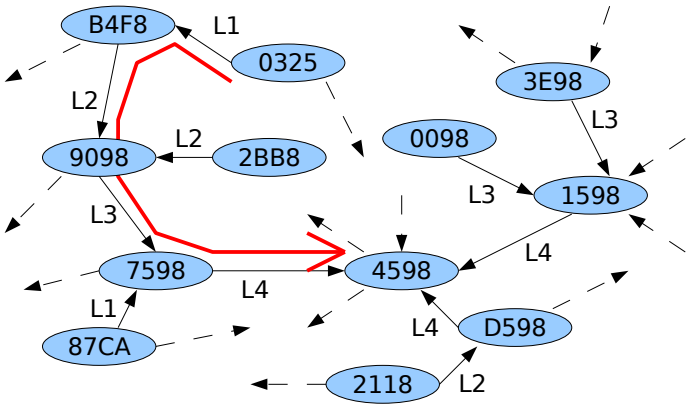

Abbildung 2: Das Plaxton „Routing-Mesh“ nach [2]

In Abbildung 2 wird ein Weg zum Knoten 4598 gesucht. Wenn wir die Suche bei 0325 starten, schaut dieser in seiner Nachbarschaftstabelle auf Level 1 (da er keine Übereinstimmung an letzter Stelle mit dem Zielknoten besitzt) an Position acht, wo er den Knoten B4F8 findet. Der zweite Knoten hat eine Übereinstimmung mit dem Zielknoten, deshalb schaut dieser direkt auf Level 2 in seine Neighbor-Map, ob er einen Knoten kennt, der eine neun an vorletzter Stelle hat. So gelangt die Anfrage zu 9098 usw.

5.3 Tapestry

Das *Overlay Netzwerk* Tapestry wird als Zwischenschicht zwischen dem Internet und einer P2P-Applikation eingesetzt. Es baut auf dem Plaxton Schema auf, erweitert dieses jedoch um die Einsatzmöglichkeit in einem dynamischen Netzwerk. Typestry ist in der Lage Netzwerkfehler zu tolerieren (*Fehlertolerantes Routing*), die bei Zulassung unzuverlässiger Knoten vorprogrammiert sind. Die Namensvergabe für einzelne Knoten gleicht der des Plaxton Schemas.
Tapestry wird bereits in verschiedenen P2P-Applikationen eingesetzt, beispielsweise in „SpamWatch“, einem Spam-Filter-Netzwerk, sowie in Bayeux, einem

Audio-/Video-Multicast-Netzwerk, und natürlich in OceanStore. Eine freie Implementierung in Java ist verfügbar[13]. Da Tapestry ein sehr komlexes und umfangreiches Thema ist, soll an dieser Stelle auf die Werke „Tapestry: A Resilient Global-Scale Overlay for Service Deployment“ [7] und „Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing“ [8] verwiesen werden, da eine ausführliche Behandlung dem Umfang dieses Artikels nicht gerecht würde.
Ein kleiner Ausschnitt, der das Bekanntmachen neuer Objekte behandelt, soll trotzdem näher betrachtet werden.

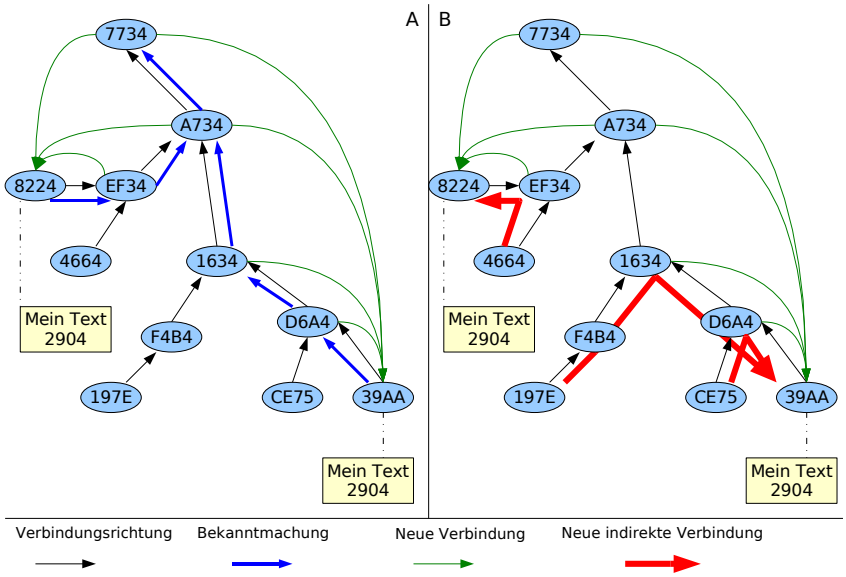

Abbildung 3: Bekanntmachen von Objekten in Tapestry nach [5]

In Abbildung 3 sehen wir, dass das Objekt „Mein Text“ einmal vom Knoten 39AA und zusätzlich als Replikat von 8224 referenziert wird. Da die Verbindungen der Knoten untereinander stets gerichtete Kanten sind, folgt auch die Bekanntmachung neuer Objekte dieser Richtung, was im linken Teil der Abbildung durch blaue Pfeile dargestellt ist. Da die Knoten 197E, F4B4, 4664 und CE75 von dieser Bekanntmachung nichts mitbekommen, müssen sie, wie im rechten Teil der Abbildung durch die roten Kanten dargestellt, erst ihre Nachbarn fragen, ob diese einen Weg kennen, um eine Verbindung aufbauen zu können.

6 Fazit und Ausblick

OceanStore befindet sich in der Entwicklung und hat definitiv noch mit Kinderkrankheiten zu kämpfen. Der Prototyp „Pond“ sieht allerdings recht vielversprechend aus, obwohl noch vieles nicht implementiert ist (Introspection)

[13] http://www.cs.ucsb.edu/~ravenben/tapestry/html/home.html

oder die Ansätze zur Realisierung noch gänzlich fehlen (floating replicas). Eine erste Testumgebung[14] ist bereits aufgebaut, 98 Rechner in 42 Institutionen in Nordamerika, Australien und Europa – wobei fraglich ist, ob diese Zahlen das angestrebte Einsatzgebiet repräsentieren.
Die Archivierungsfunktion sieht bereits sehr vielversprechend aus, was OceanStore für einen Einsatz als riesiges Datenlager interessant macht. Jedoch muss erwähnt werden, dass die Möglichkeit GroupWare-Daten zu speichern, wie in der Einleitung angesprochen, wenig Akzeptanz finden wird. Diese Daten werden im Allgemeinen sehr vertraulich behandelt und das Speichern auf einem öffentlich zugänglichen Netzwerk stellt nicht das Optimum im Umgang mit vertraulichen Daten dar.
Die in „Pond: the OceanStore Prototype“ [4] veröffentlichten Leistungsanalysen zeigen, dass OceanStore gut bezüglich der Datenmenge und der Benutzeranzahl skaliert. Dies wird offensichtlich durch die ausgeglichene Verteilung der Daten und somit der Last auf teilnehmende Server erreicht. Offene Fragen bleiben jedoch, als Beispiel die *Quality of Service*. Wie soll diese garantiert werden, wenn das System auf verschiedene Unternehmen verstreut ist?
Konkurrenzsysteme machen im Vergleich zu OceanStore viele Einschränkungen, um den Aufwand zu minimieren. So erlaubt „PAST“ das Ändern vorhandener Daten nicht oder das Cachen ist nicht überall erlaubt (zB. bei NFS). Auch die fehlende Vertrauenswürdigkeit wird nicht von allen Systemen berücksichtigt, was einen Einsatz nur in bestimmten Umgebungen zuläßt. Eine Garantie für das Auffinden tatsächlich vorhandener Daten wird, wie in Abschnitt 5 erwähnt, auch nicht von allen Systemen gegeben.
Um die Akzeptanz und das Interesse zu wecken, wurde bereits eine OceanStore-API[15] veröffentlicht, die eine Integration in bestehende UNIX-Betriebssysteme ermöglichen soll. Auch ein Browser-Plugin soll erhältlich sein, das einen Zugang zum OceanStore Netzwerk gewährt.

[14] http://oceanstore.cs.berkeley.edu/publications/talks/tahoe-2003-01/kubitron_overview.pdf
[15] Application Programming Interface

Literatur

[1] J. Bloemer, M. Kalfane, R. Karp, M. Karpinsiki, M. Luby, and D. Zuckerman. An xor-based erasure-resilient coding scheme. *Technical Report, International Computer Science Institute, Berkeley, California*, 1995.

[2] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[3] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An extremely wide-area storage system. *U.C. Berkeley Technical Report UCB//CSD-00-1102*, 1999.

[4] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: the oceanstore prototype. *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)*, 2003.

[5] S. Rhea and J. Kubiatowicz. Probabilistic location and routing. *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2002.

[6] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. *ACM SOSP*, 2001.

[7] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 2004.

[8] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *U.C. Berkeley Technical Report UCB//CSD-01-1141*, 2000.