Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Thesis Outline

Mykhaylo Kabalkin

Persistenz-Service System

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Mykhaylo Kabalkin

Persistenz-Service System

Thesis Outline eingereicht im Rahmen des Seminar-Ringvorlesung
im Studiengang Informatik Master of Science
am Studiendepartment Informatik
des Departments Informations- und Elektrotechnik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuer : Prof. Dr. Kai von Luck

Abgegeben am 14. Februar 2007

Mykhaylo Kabalkin

Thema der Thesis Outline

Persistenz-Service System

Stichworte

Persistenz-Service System, Lustre File System, Git-Fast Version Control

Kurzzusammenfassung

In diesem Dokument beschreibt der Autor seine ersten Ideen und Entwürfe für seine Masterarbeit. Die Technologien, die dabei seines Erachtens verwendet werden können, und die Zusammenhänge dazwischen werden vorgestellt.

Mykhaylo Kabalkin

Title of the paper

Persistence-service system

Keywords

Persistence-Service System, Lustre File System, Git-Fast Version Control

Abstract

In this paper the author describes his first ideas and drafts for his masterthesis. The technologies which can be used, besides, of his opinion, and the connections in between will be introduced.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Motivation, Problematik und Ziele

Im Rahmen der Vorlesung „Anwendungen I“ setzte sich der Autor dieser Ausarbeitung mit dem Thema „Gemeinsamer Speicher in Collaborative Workspace“ [Kabalkin (2006)] auseinander. Die dabei gewonnenen Erkenntnisse gellten als Grundlagen für die Vorlesungen „Seminar-Ringvorlesung“, in deren Rahmen diese Ausarbeitung erstellt wird.

Heutige Software-Systeme bieten riesige Funktionalität, aber die Anforderungen steigen immer noch. Während der Mitarbeit im Projekt „Collaborative Workspace“ stellte der Autor dieser Ausarbeitung fest, dass mehrere Fragen bei der Verwendung von solchen Software-Systemen noch offen sind. Die Benutzer eines System müssen sich immer Gedanken über den Speicherplatz machen. Wo und wie die Daten eines Benutzers gespeichert, wie sie wieder gefunden und geladen werden, ist auch der Punkt, um den sich der Benutzer selbst kümmern soll. Sind die Daten immer verfügbar, wenn der Benutzer sie braucht, ist auch eine offene Frage. Was geschieht, wenn der Rechner ausfällt? Ist das System ausfallsicher? Oft braucht ein Benutzer die Möglichkeit, auf seine alten Daten zuzugreifen. Es ist bei vielen heutigen Systemen auch nicht möglich. Sicherlich kommen im Laufe der Masterarbeit weitere Fragen hoch, die auch mit Hilfe des Systems geklärt werden sollten. Das zu entwerfende System ist nicht für Homeanwender gedacht, sondern für die große Rechenzentren oder Universitäten.

Aus der oben beschriebenen Problematik setzt sich der Autor den Entwurf eines Persistenz-Service Systems als Ziel seiner Masterarbeit. Mit dem Persistenz-Service System sollen Benutzer alle oben gestellten Fragen klären können. Dies sollte als Voraussetzung für das System gelten. Die genauen Anforderungen an das Persistenz-Service System werden im Kapitel 2 definiert. In dieser Ausarbeitung werden erste Ideen und Entwürfe des Autors für die Masterarbeit beschrieben. Die Technologien, die dabei seines Erachtens verwendet werden können, und die Zusammenhänge dazwischen werden vorgestellt. Einige Komponenten des System werden detaillierter beschrieben, andere müssen noch untersucht werden. Das Zusammenspiel einiger Komponenten muss im gesamten System noch überprüft werden. Auch Risiken, die bei der Verwendung der einen oder anderen Technologie im gesamten Verlauf der Masterarbeit entstehen können, werden im Kapitel 4.2 aufgelistet.

2 Anforderungen an das System

In diesem Kapitel werden die genauen Anforderungen an das verteilte Persistenz-System beschrieben. Die möglichen Einsätze, mit denen diese Anforderungen beantwortet werden können, werden kurz vorgestellt.

Zwei der wichtigen Aspekte eines System sind die Ausfallsicherheit des System und die Verfügbarkeit von Daten. Sollte ein Rechner ausfallen, soll dies für den Benutzer transparent sein, und er kann weiter mit seinen Daten arbeiten, als ob es nicht passiert wäre. Das System muss trotz des möglichen Ausfalls von Servern oder des Verlustes von Nachrichten korrekt arbeiten. Diese Funktionalität kann man durch die Redundanz von Daten gewinnen.

Da das System für große Rechenzentren oder Universitäten gedacht ist, spielt die Skalierbarkeit eine große Rolle. Dies kann durch die Verteilung von Daten erreicht werden. Hier handelt es sich nicht um eine Verteilung auf mehrere Festplatten, sondern um die Verteilung der Daten auf mehrere Rechner.

Oft wollen die Benutzer auf ihren alten Datenbestand zugreifen, obwohl die Daten schon oft geändert wurden (z.B. möchte man mit den Daten vom 01. Januar 2005 arbeiten). Auch die Verfolgung von Änderungen kommt meistens in Frage. Hier könnte eine Versionskontrolle zum Einsatz kommen.

In einem Persistenz-Service System muss Sicherheit gewährleistet werden. Ein vernünftiges Sicherheitsmodell kann sie garantieren.

Die Daten sollten zugriffstransparent bleiben, gleichgültig, von welchem Host der Benutzer auf sie zugreift. Mehrere Benutzer könnten im gesamten System gleichzeitig mit denselben Daten arbeiten. Die Konsistenz der Daten ist auch ein wichtiger Punkt in einem Software-System. Es sollte möglich sein, die Daten zu ändern, ohne Kenntnisse zu haben, wie viele Kopien der Daten in dem gesamten System existieren. Die Benutzer sollten auch keine Kenntnisse über die Örtlichkeit der nicht lokalen Kopien haben.

In einem Softwaresystem spielt die Performanz eine sehr große Rolle. Trotz variierender Lasten am File-Server müssen die Clients mit genügender Geschwindigkeit arbeiten können.

Im nächsten Kapitel beschreibt der Autor den ersten Entwurf der Systemarchitektur und deren Komponenten im Einzelnen.

3 Mein Ansatz

Die Abbildung 3.1 stellt den ersten Architekturentwurf des Autors für das Persistenz-Service System. Das ganze System wird in einzelne Module unterteilt. Da alle Module voneinander unabhängig bleiben, kann das laufende System auf einzelne Module verzichten.

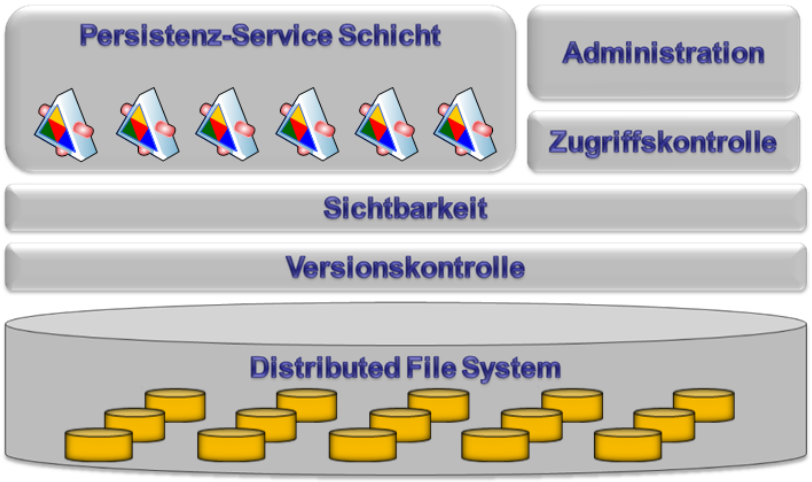

Abbildung 3.1: Erster Entwurf einer Architektur des Persistenz-Service Systems.

In diesem Abschnitt werden die Module einzeln beschrieben. Dabei geht der Autor bezüglich der Systemarchitektur nach der Bottom-up Methode vor und präsentiert auch einige Technologien, die eingesetzt werden könnten.

3.1 Distributed File System

Da es hier um die Verteilung über mehrere Rechner geht, kommt der Einsatz von Redundant Array of Independent Disks (RAID) nicht in Frage. RAID-Systeme dienen zur Organisation mehrerer Festplatten zu einem logischen Laufwerk und sind nur für lokale die Speicherung gedacht. Dagegen kann die Verteilung über mehrere Rechner durch Einsatz eines verteilten

Dateisystems erreicht werden. In einem verteilten Dateisystem sind die Daten immer verfügbar. Sollte ein Rechner ausfallen, ist dies für Endbenutzer transparent. In [Ramamurthy (2004)] definiert Dr. Bina Ramamurthy ein verteiltes Dateisystem wie folgt:

> „Distributed file systems support the sharing of information in the form of files throughout the intranet. A distributed file system enables programs to store and access remote files exactly as they do on local ones, allowing users to access files from any computer on the intranet."

Als Beispiele für verteilte Dateisysteme können Google File System [Gobioff u. a. (2003)], Andrew File System [Nydick u. a. (2001)] und Lustre File System [Cluster (2002)] genannt werden. Der Autor dieser Ausarbeitung wählte das Lustre File System als verteiltes Dateisystem für das gesamte Persistenz-Service System aus. Die Gründe dafür sind in [Kabalkin (2007b)] beschrieben.

3.1.1 Lustre File System

Das Lustre File System (im Folgenden Lustre FS genannt) ist ein skalierbares, sicheres, robustes und ausfallsicheres Cluster Datei System, und wurde von Cluster File System Inc. entwickelt. Das Luster FS besteht aus drei Hauptkomponenten, deren Interaktion in der Ab-

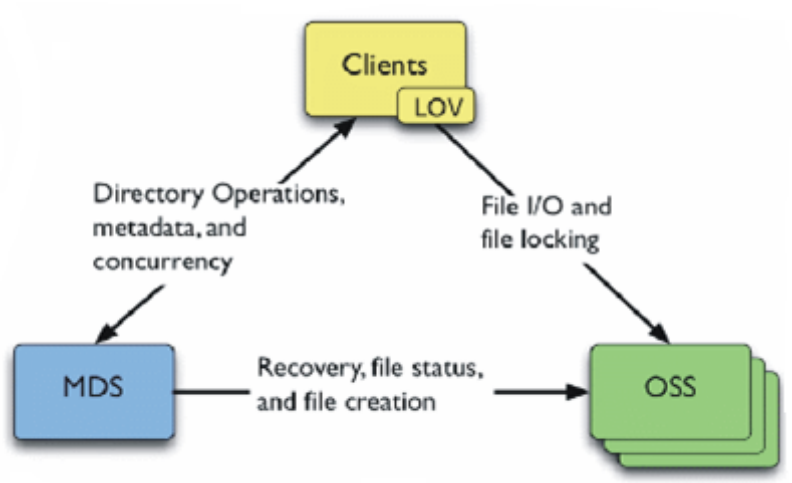

Abbildung 3.2: Interaktion zwischen Systemen in Lustre File System

bildung 3.2 dargestellt ist:

- **Meta Data Server (MDS)** bietet Back-End Speicher für Metadaten Service, speichert Referenzen zu echten Daten und aktualisiert diesen Service bei jeder Transaktion über

die Netzwerkschnittstelle. Das Lustre FS beinhaltet geclusterte Metadaten. Die Bearbeitung von Metadaten wird mit Hilfe von der Lastverteilung, die zur Folge hat, dass der gleichzeitige Zugriff auf die Metadaten sehr komplex ist.

- **Object Storage Server (OSS)** kann mehrere Netzwerkschnittstellen und normalerweise eine oder mehrere Festplatten haben. OSS bietet Datei Input/Output Service und speichert echte Daten.
- **Lustre Clients** interagieren mit Object Storage Servern für Daten I/O und mit dem Meta Data Server für den Metadatentransfer.

Ein großer Vorteil vom Lustre FS ist die Trennung zwischen Metadaten und echten Daten, die das System mit sich bringt. Die Datenredundanz kann man durch Replikationen erreichen.

3.1.2 Fazit

Durch den Einsatz von Lustre FS gewinnt das Persistenz-Service System an Ausfallsicherheit und Skalierbarkeit. Die Verfügbarkeit der Daten wird auch garantiert. Im Grunde genommen ist das Lustre FS nach außen als ein normales Dateisystem sichtbar. Es bietet aber zusätzliche Funktionalität, mit der die im Kapitel 2 definierten Anforderungen im Persistenz-Service System beantwortet werden könnten.

3.2 Versionskontrolle

Durch die Versionskontrolle können die Änderungen der Daten verfolgt werden. Es kann gefragt werden, ob der Zugriff auf alle Versionen eines Dokumentes gegeben werden sollte und alle Änderungen verfolgt werden sollten. Bei dem Persistenz-Service System ist es kein wichtiges Argument für den Einsatz von der Versionskontrolle. Ein wichtiges Argument dafür ist es, die Zeitschlitze des gesamten Systems oder eines Teiles davon zurücksetzen zu können (z.B. möchte man den Stand der Daten vom 01.01.2000 sehen und damit arbeiten). Hier muss berücksichtigt werden, dass der Zeitpunkt, in dem eine und/oder die letzte Replikation der Daten erzeugt wurde, nicht relevant ist. Die zu betrachtende Zeit ist die Zeit, wenn ein Benutzer seine Änderungen abspeichert. Wie lange für die Replikation der Daten gebraucht wird, sollte hier irrelevant bleiben.

Die Überlegungen, dass ein Benutzer auf seinen alten Datenbestand zugreifen kann, verfolgt die Firma Apple Computer, Inc. mit ihrem neuen Betriebssystem Mac OS X Leopard. Mit der Time Mashine [Apple] bietet Apple Computer, Inc. den Benutzern die Möglichkeit, auf ihre alten Daten zuzugreifen, obwohl die Daten möglicherweise schon gelöscht wurden.

3.2.1 Concurrent Versions System

Concurrent Versions System (CVS) ist ein Konfigurationsmanagementsystem zur Versionsverwaltung von Dateien und wird in der Software Entwicklung für die Verwaltung von Quellcode eingesetzt. CVS ermöglicht Revisionsverwaltung für Projekte und arbeitet auf dem gesamten Verzeichnisbaum. CVS verwendet ein zentrales Repository. Das Repository ist ein vom Versionskontrollsystem und dessen Administrator gepflegter zentraler Ablageort für alle Daten eines Projektes. Das Repositry enthält alle Angaben, die für die Wiederherstellung von früheren Versionen einer Projektdatei notwendig sind. CVS verwendet ein Verfahren zum Zusammenführen (merging) von Daten, das jederzeit den gleichzeitigen Zugriff auf alle Dateien sowie die parallele Arbeit aller Beteiligten erlaubt. Die Unterschiede zwischen bestimmten Versionen eines Dokumentes können herausgestellt werden. Unterschiedliche Versionen können verglichen und gemergt werden. Durch CVS wird die Koordination der Arbeit erleichtert.

3.2.2 Distributed Concurrent Versions System

CVS ist stabil, erprobt und seine Stärken und Schwächen hinlänglich bekannt. Wird die Projektarbeit auf verschiedene Standorte verteilt, sind jedoch schnell die Grenzen von CVS erreicht. Das Distributed Concurrent Versions System (DCVS) ist ein Versionskontrollsystem, das den Projektbeteiligten ermöglicht, an einem Projekt leistungsfähig zusammenzuarbeiten, obwohl sie auf unterschiedlichen Standorten verteilt arbeiten. DCVS ersetzt das streng zentralistische Modell von CVS durch eine Vielzahl gleichberechtigter Repositories. Dies ist der Hauptunterschied zu CVS. Ein Main-Repository kann definiert werden. Lokale Repositries können bei Notwendigkeit gegen das Main-Repository automatisch oder manuell synchronisiert werden.

Ein Vertreter des Distributed Concurrent Versions Systems ist das Git - Fast Version Control System (Git) [Torvalds], das ursprünglich für die Verwaltung des Linux-Kernels entwickelt wurde. Jedes Git Arbeitsverzeichnis ist ein selbständiges Repository mit voller Kontrolle und dem Verfolgen aller Änderungen unabhängig vom Netzzugang oder einem zentralen Server. Jeder Benutzer besitzt eine lokale Kopie des gesamten Repository. Es wird nicht zwischen lokalen und entfernten Repositories unterschieden. Der Datentransfer zwischen Repositories ist bei Git flexibel. Die Daten können mit einer Reihe verschiedener Protokolle zwischen Repositories übertragen werden. Git besitzt ein sehr effektives eigenes Protokoll, das auf TCP-Port 9418 arbeitet. Ebenso kann der Transfer über SSH oder (weniger effizient) über HTTP, HTTPS oder FTP erfolgen. Im Gegensatz zu traditionellen Versionskontrollsystemen wie CVS arbeitet Git nicht auf einzelnen Dateien, sondern auf Verzeichnissen bzw. Hierarchien von Verzeichnissen.

3.2.3 Fazit

Das Git - Fast Versions Control System wurde von dem Autor dieser Ausarbeitung zum Einsatz im gesamten Persistenz-Service System gewählt. Ein großer Vorteil von Git ist seine Effizienz bei großen Projekten. Git ist schnell und skaliert gut im Umgang mit großen Projekten und langen Repositories. Es ist bei den meisten Aktionen und Zugriffen eine Größenordnung schneller als traditionelle Versionskontrollsysteme.

In dem Persistenz-Service System sollte die Versionierung der Daten durch den Benutzer konfigurierbar sein. Jeder Benutzer könnte festlegen, welche (alle oder nur bestimmte) Daten versioniert werden sollten. In dem ersten Prototyp wird das Persistenz-Service System solche Funktionalität nicht anbieten. Erstmal werden alle Daten versioniert, und ein Benutzer hat keine Kontrolle darüber.

3.3 Zugriffskontrolle und Sichtbarkeit der Daten

Die Sicherheit spielt in einem Software-System eine große Rolle. Wer darf auf welche Daten zugreifen? Welche Daten sind für welchen Benutzer sichtbar? Solche Fragen werden im einem Multiusersystem immer gestellt. Es ist wichtig, die Sicherheitsbelange der Benutzer zu erkennen und zu gewährleisten. Dies kann mit einer geeigneten Sicherheitsstrategie umgesetzt werden.

Als langfristiges Ziel wird von dem Autor ein rollenbasiertes Sicherheitmodell im Persistenz-Service System gesetzt. Dies wird aber nicht im Rahmen des ersten Prototyps entworfen. Hier ist der Entwurf eines Zugriffsmatrix-Sicherheitmodelles angedacht.

3.3.1 Zugriffsmatrix Sicherheitsmodell

Das Zugriffsmatrix-Sicherheitsmodell ist das einfachste und älteste Sicherheitsmodell. Durch die Zugriffsmatrix werden die Zugriffsrechte von Subjekten auf Objekte modelliert. Es ist möglich, Objekte und Subjekte in Bezug auf die Anforderungen der Anwendung festzulegen. Die Zugriffsrechte können allgemein oder objektspezifisch modelliert werden. Man unterscheidet Zugriffsmatrix-Modelle mit statischer bzw. dynamischer Zugriffsmatrix. Bei einer statischen Zugriffsmatrix sind die Änderungen der Zugriffsrechte nicht möglich. Dies ist zur Modellierung der Anwendungen geeignet, in denen der Rechtezustand von Anfang an festgelegt ist und über lange Zeit nicht geändert wird. Bei einer dynamischen Zugriffsmatrix werden Zustandsübergänge (Veränderungen der Zugriffsmatrix) durch die Ausführung von Befehlen abgebildet, die durch eine Folge von Elementaroperationen spezifiziert werden.

Sechs Elementaroperationen zum Erzeugen bzw. Löschen von Objekten und Subjekten sowie zur Rechteweitergabe und - rücknahme sind in [Harrison u. a. (1976)] definiert.

In dem Persistenz-Service System sind zunächst folgende Zugriffsrechte vorgesehen:

- Lesen von Daten
- Suchen von Daten
- Erzeugen von neuen Daten
- Editieren von Daten (d.h. Erzeugen einer neuen Version von schon existierenden Daten)
- Rechtevergabe

Setzt man ein verteiltes Dateisystem ein, kommt noch eine weitere wichtige Frage dazu: Wie werden die Daten gelöscht? Das Problem dabei besteht darin, dass es in vielen verteilten Dateisystemen nicht möglich ist, die Daten zu löschen. In diesem Fall kann man die Daten, die gelöscht werden sollen, als gelöscht markieren oder durch Zugriffskontrolle allen Benutzern dieser Daten keine Leserechte mehr geben.

3.3.2 Sicherheit im Lustre File System

Die Sicherheit des Dateisystems ist ein sehr wichtiger Aspekt eines verteilten Dateisystems. Die Standardaspekte der Sicherheit sind Authentifikation, Autorisation und Verschlüsselung. Das Lustre File System unterstützt Object Storage Server mit Network Attached Secure Disks (NASD) [NASD].

Das Lustre FS kann mit einem existierendem Authentifikationmechanismus einfach integriert werden. Das Authentifikationmechanismus benutzt Generic Security Service Application Programming Interface (GSS-API). Das GSS-API ist ein offener Standard, der Sessionmanagement, Authenifizierung, Datenintegrität und Vertraulichkeit von Daten zur Verfügung stellt. Die Kerberos 5 [RFC4120] und PKI Mechanismen werden als Backend für die Authentifizierung in dem Lustre FS benutzt. Der Ablauf eines Lesezugriffes in dem Lustre File System ist in der Abbildung 3.3 dargestellt.

In den nächsten Versionen wird das Lustre File System die Autorisierung durch Access Control Lists (ACL) unterstützen, die der POSIX[1] ACL Semantik folgen. Die Flexibilität und zusätzliche durch ACLs zur Verfügung gestellte Fähigkeiten sind in Clustern besonders wichtig, weil auf Grund dessen Tausende von Knoten und Benutzerkonten unterstützt werden können.

[1] POSIX - portabeles Betriebsysteminterface, das auf UNIX basiert

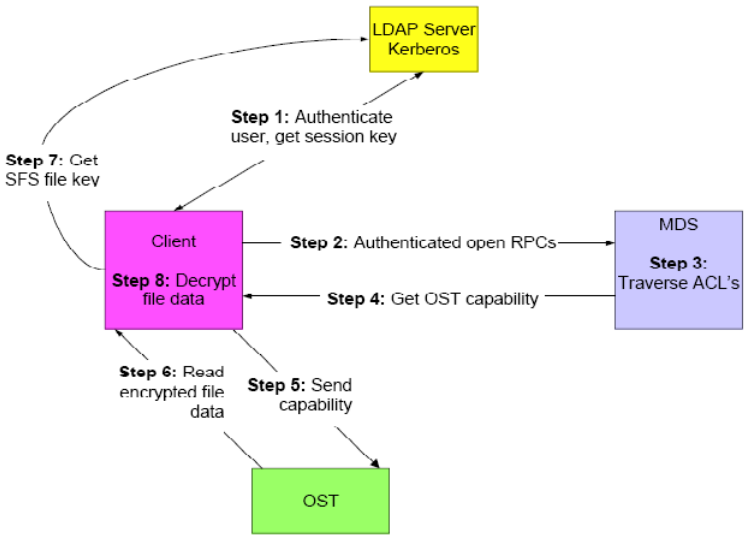

Abbildung 3.3: Lustre Read File Security

3.3.3 Fazit

Das Lustre File System bringt viele Sicherheitsmechanismen mit sich. Die Konfiguration dieser Mechanismen muss noch untersucht und evaluiert werden.

3.4 Persistenz-Service Schicht

Die Persistenz-Service Schicht ist für die Schnittstelle nach aussen verantwortlich. Es ist sehr wichtig, eine einheitliche Schnittstelle zu definieren, die das gesamte System nach aussen zur Verfügung stellt. Gegen diese Schnittstelle können weitere Dienste und Anwendungen (z.B. Suchdienst, Abodienst, Maus Pointer) an das Persistenz-Service System angebunden werden. Der Autor wird nicht nur eine einfache Schnittstelle, sondern auch das gesamte Persistenz-Service System als einen Verzeichnisbaum zur Verfügung stellen.

4 Ausblick

4.1 Vorgehensweise

Wie schon in dieser Ausarbeitung erwähnt wurde, sind alle Systemmodule voneinander unabhängig. Aus diesem Grund können sie in beliebigen Kombinationen eingesetzt werden, ohne dass sich der Entwickler Sorgen machen muss, dass das gesamte Persistenz-Service System in solch einer Kombination von Komponenten nicht funktioniert. Ein großer Vorteil dabei ist es, dass ein einfaches Application Programming Interface (API) nach aussen zur Verfügung gestellt werden kann, obwohl das gesamte Persistenz-Service System noch nicht vollständig entworfen ist. Das API muss dabei als eine Schnittstelle nach aussen gleich aussehen, gleichgültig, welche Kombinationen von Komponenten den Benutzern zur Verfügung gestellt werden.

4.2 Risiken

Der Autor dieser Thesis Outline stellte schon bei seinen ersten Überlegungen einige Risiken fest, die bei dem Entwurf des Persistenz-Service Systems berücksichtigt werden müssen.

Da das Persistenz-Service System aus vielen Komponenten bestehen wird, könnte die Konfiguration des gesamten Systems sehr komplex werden. Es ist erwünscht, dass das System auf IBM Blade Servern [Blade] laufen sollte. Dies ist auch ein weiteres Risiko. Die Problematik mit den IBM Blade Servern, die der Autor bei der Mitarbeit in dem Projekt „Collaborative Workspace“ hatte, wird in dem Projektbericht [Kabalkin (2007a)] beschrieben.

Ob das Git - Fast Versions Control System auf das Lustre File System aufgesetzt werden kann, ist noch zu evaluieren. Auch wenn es möglich ist, bleibt die Frage, ob sich die Versionskontrolle auf einem verteilten Dateisystem gleich wie auf einem Standard-Dateisystem verhält, noch offen.

Literaturverzeichnis

[Blade] : *Blade Center*. – URL http://www-03.ibm.com/systems/de/bladecenter/

[NASD] : *Network Attached Secure Disks (NASD)*. – URL http://www.pdl.cmu.edu/NASD/

[RFC4120] : *RFC4120, The Kerberos Network Authentication Service (V5)*. – URL http://tools.ietf.org/html/rfc4120

[Apple] APPLE, Computer: *Time Mashine in Mac OS X Leopard*. – URL http://www.apple.com/macosx/leopard/timemachine.html

[Cluster 2002] CLUSTER, Filesystem: *Lustre: A Scalable, High-Performance File System*. November 2002. – URL http://www.lustre.org/docs/whitepaper.pdf

[Eckert 2004] ECKERT, Claudia: *IT-Sicherheit*. München [u.a.] : Oldenbourg, 2004. – ISBN 3-486-20000-3

[Gobioff u. a. 2003] GOBIOFF, Howard ; LEUNG, Shun-Tak ; GHEMAWAT, Sanjay: *The Google File System*. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles. Bolton Landing, New York, USA. October 2003. – URL http://labs.google.com/papers/gfs-sosp2003.pdf

[Harrison u. a. 1976] HARRISON, Michael A. ; RUZZO, Walter L. ; ULLMAN, Jeffrey D.: Protection in operating systems. In: *Commun. ACM* 19 (1976), Nr. 8, S. 461–471. – ISSN 0001-0782

[Kabalkin 2006] KABALKIN, Mykhaylo: *Gemeinsamer Speicher in Collaborative Workspace*. Juli 2006. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2006/kabalkin/abstract.pdf

[Kabalkin 2007a] KABALKIN, Mykhaylo: *Lustre File System*. Februar 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master06-07-proj/kabalkin/paper.pdf

[Kabalkin 2007b] KABALKIN, Mykhaylo: *Verteilte Dateisysteme*. Februar 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master06-07-aw/kabalkin/abstract.pdf

[Miller u. a.] MILLER, Brad ; OLSON, Curtis ; TRUCKENMILLER, Dave: *Distributed Concurrent Version System*. – URL citeseer.ist.psu.edu/article/miller93distributed.html

[Nydick u. a. 2001] NYDICK, Daniel ; BENNINGER, K. ; BOSLEY, B. ; ELLIS, J. ; GOLDICK, J. ; KIRBY, C. ; LEVINE, M. ; MAHER, C. ; MATHIS, M.: *An AFS-based mass storage system at the Pittsburgh Supercomputing Center*. Eleventh IEEE Symposium. October 2001

[Ramamurthy 2004] RAMAMURTHY, Bina: *Destributed File Systems*. September 2004. – URL http://www.cse.buffalo.edu/gridforce/fall2004/DistributedFileSystemSept29.pdf

[Torvalds] TORVALDS, Linus: *Git - Fast Version Control System*. – URL http://git.or.cz