Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung

Mykhaylo Kabalkin

Distributed File System

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Mykhaylo Kabalkin

Distributed File System

Ausarbeitung eingereicht im Rahmen der Vorlesung Anwendungen II
im Studiengang Informatik Master of Science
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuer : Prof. Dr. Kai von Luck

Abgegeben am 21. Februar 2007

Mykhaylo Kabalkin

Thema der Ausarbeitung

Distributed File System

Stichworte

Verteilte Dateisysteme, Network File System, Andrew File System, Google File System, BitTorrent, Lustre File System

Kurzzusammenfassung

Verteilte Dateisysteme unterstützen die Verteilung der Informationen in Form der Dateien überall im Intranet. Ein verteiltes Dateisystem erlaubt Programmen, die Dateien entfernt zu speichern und auf diese entfernt zuzugreifen, als ob sie lokal wären, und Benutzern auf Dateien von beliebigen Rechnern im Intranet zuzugreifen.

Mykhaylo Kabalkin

Title of the paper

Distributed file system

Keywords

Distributed File System, Network File System, Andrew File System, Google File System, BitTorrent, Lustre File System

Abstract

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.

Inhaltsverzeichnis

Tabellenverzeichnis **5**

Abbildungsverzeichnis **6**

1 Einführung **7**
1.1 Motivation 7
1.2 Vision 7

2 Charakteristiken verteilter Dateisysteme **9**
2.1 Transparenz 9
2.1.1 Zugriffstransparenz 9
2.1.2 Mobility Transparenz 9
2.1.3 Ortsunabhängigkeit 9
2.1.4 Performanz Transparenz 9
2.1.5 Scaling Transparenz 10
2.2 Redundanz von Daten 10
2.3 Konsistenz von Daten 10
2.4 Fehlertoleranz und Sicherheit 10
2.5 Hardware und Betriebssystem Heterogenität 10

3 Bekannte verteilte Dateisysteme **11**
3.1 Network File System 11
3.2 Andrew File System 12
3.3 BitTorrent 12
3.4 Google File System 14
3.5 Lustre File System 16

4 Zusammenfassung **18**

Literaturverzeichnis **19**

Tabellenverzeichnis

4.1 Vergleich verteilter Dateisysteme. 18

Abbildungsverzeichnis

3.1 The basic NFS architecture for UNIX systems 11
3.2 BitTorrent Architektur 13
3.3 Google File System Architektur [Gobioff u. a. (2003), S.3] 15
3.4 Schreibkontrolle und Datenfluss in GFS [Gobioff u. a. (2003), S.5] 16
3.5 Interaktion zwischen Systemen in Lustre File System 17

1 Einführung

1.1 Motivation

Heutige netzorientierte Rechenumgebungen verlangen hoch-perfomante, network-aware Dateisysteme, die sowohl mehr Speicher von individuellen Systemen als auch die Verteilung von Daten von kooperativen Systemen voraussetzen. In einem solchen System ist es sehr wichtig, dass auf Daten schnell zugegriffen werden kann. Solche Systeme sollten natürlich ausfallsicher sein. In der Softwareentwicklung spielen die Kosten heutzutage auch eine Rolle.

Aus diesem Grund ist der Autor dieser Ausarbeitung auf der Suche nach einem kostengünstigen, server-schnellen und ausfallsicheren verteilten Dateisystem. In diesem Dokument werden die Anforderungen an solchen Systemen beschrieben. Einige der bekannten verteilten Dateisysteme, die eingesetzt werden könnten, werden bezüglich der Anforderungen verglichen und dargestellt.

Die im Rahmen von dieser Ausarbeitung gewonnenen Erkenntnisse werden als Grundlagen für das Projekt „Collaborative Workspace“ [Kabalkin (2007a)] gelten. Ein dabei ausgewähltes verteiltes Dateisystem wird im Rahmen des Projektes detaillierter untersucht.

1.2 Vision

Ein Dateisystem (File System) wird als ein Ordnungs- und Zugriffssystem für Daten definiert, die auf einem Datenträger gespeichert sind. Das Dateisystem ist ein Bestandteil eines Betriebssystems, und ein Betriebssystem ist ebenfalls in einem Dateisystem gespeichert. Einige bekannte Dateisysteme sind Third extended file system (Ext3); Fourth extended file system (Ext4), seit Version 2.6.19 offizieller Bestandteil des Linux-Kernels; das Berkeley Fast File System wird in verschiedenen BSD-Derivaten (FreeBSD, NetBSD und OpenBSD) sowie in Solaris und NextStep verwendet; File Allocation Table (FAT12, FAT16 und FAT 32); Virtual FAT (VFAT), das die längeren Dateinamen unterstützt; New Technology File System (NTFS); das erste Apple Dateisystem (Apple DOS); Hierarchical File System (HFS), das mit dem Mac OS entwickelt wurde.

Ein verteiltes Dateisystem hat eine ähnliche Aufgabe wie ein Standard-Dateisystem in konventionellen Betriebssystemen und erlaubt den Zugriff auf Dateien in anderen, entfernten Rechnern. Verteilte Systeme werden zur Realisierung von anderen Diensten benutzt (z.B. Persistentes Speichern für Transaktionssysteme, Namensserver, Authentication-Server, usw.).

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.“

Dieses Zitat betrachtet der Autor als Vision seiner Arbeit.

2 Charakteristiken verteilter Dateisysteme

2.1 Transparenz

2.1.1 Zugriffstransparenz

Jeder Benutzer kann von jedem beliebigen Host auf seine Daten zugreifen. Die Verteilung der Daten sollte den Benutzern transparent bleiben.

2.1.2 Mobility Transparenz

Clientprogramme und Administrationsdienste sollten sich nicht ändern, wenn die Daten von einem zu einem anderen Speicherplatz verschoben werden.

2.1.3 Ortsunabhängigkeit

Daten eines Benutzers könnten auf verschiedenen Hosts vorliegen. Clientprogramme sollten mit den festgelegten Namespaces arbeiten, und Namen der Dateien sollten konsistent bleiben, gleichgültig, wo die Daten aktuell vorliegen und von wo aus auf die Daten zugegriffen wird. Wird eine Datei im System verlagert, so soll das den Zugriff nicht beeinträchtigen.

2.1.4 Performanz Transparenz

Mehrere Benutzer können gleichzeitig auf dieselben Daten zugreifen. Trotz variierender Lasten am File-Server sollten die Clients mit genügender Geschwindigkeit arbeiten können.

2.1.5 Scaling Transparenz

Ein Dienst kann durch das zusätzliche Wachstum erweitert werden, um einer große Neztlast bewältigen zu können.Reihe von Last- und Netzgrößen zu handeln.

2.2 Redundanz von Daten

Dieselben Daten könnten im System mehrfach repliziert vorliegen. Dies sorgt für bessere Datenzugriffskapazität und bessere Fehlertoleranz.

2.3 Konsistenz von Daten

Multiple, parallele Zugriffe auf die Daten sollten konsistente Daten repräsentieren, d.h. der Unterschied zwischen Speicherorten der Daten oder die Wartezeit des Updates der Daten sollten nicht zu unterschiedlichen Repräsentationen von Daten führen. Die Metadaten sollten an alle Clients konsistent repräsentiert werden.

2.4 Fehlertoleranz und Sicherheit

Fehlertoleranz ist eine wichtige Anforderung, wenn große Datenmengen transportiert werden. Das Dateisystem sollte trotz des möglichen Ausfalls von Servern oder des Verlustes von Nachrichten korrekt arbeiten. Transiente Kommunikation sollte auf keinen Fall zu der Korruption der Daten führen. Clientabfragen sollten authentifiziert und Datentransfer sollte verschlüsselt werden.

2.5 Hardware und Betriebssystem Heterogenität

Dienste, die angeboten werden, sollten nicht voraussetzen, dass die Clients spezielle Hardware und Betriebssysteme benötigen, um mit dem System arbeiten zu können.

3 Bekannte verteilte Dateisysteme

3.1 Network File System

Bereits 1985 wurde das Network File System von Sun Microsystem als erstes verteiltes Dateisystem eingeführt. Seit 1989 steht die Spezifikation des Systems auch anderen Herstellern zur Verfügung. Das Hauptziel des Systems ist es, einen transparenten Zugriff auf Dateien in lokal verteilten Systemen zu gewährleisten. Mittlerweile existiert schon die NFS Version 4 Protocol Specification [Noveck u. a. (2003)]. Die in der Abbildung 3.1 vorgestellte NFS Architektur ist die klassische Client-Server Architektur. Ein Server kann Verzeichnisse exportieren/freigeben, verwaltet Zugriffsrechte und ist zustandslos. Ein Client kann dagegen die vom Server freigegebenen Verzeichnisse importieren (mounten). Diese erscheinen dann wie lokale Verzeichnisse und können so genutzt werden. Bei dem Zugriff verbleiben die Dateien auf dem Server, auf dem auch die Operationen ausgeführt werden. Da ein Cli-

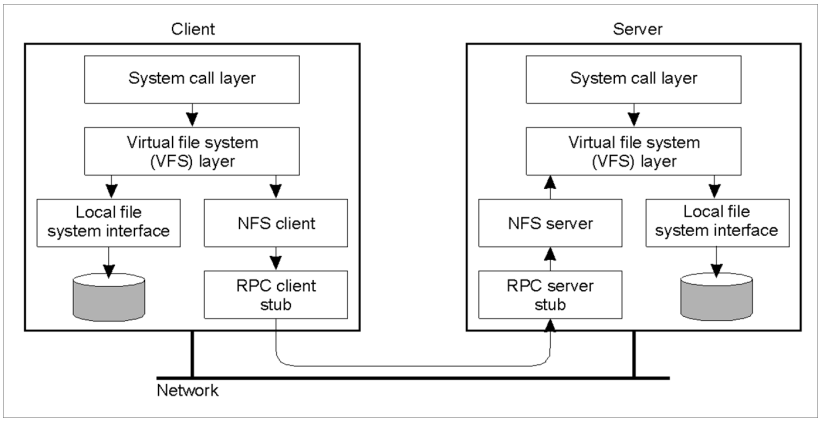

Abbildung 3.1: The basic NFS architecture for UNIX systems

ent die Stellen in seinem Namensbaum festlegt, wo das zusätzliche File-System gemountet wird, gibt es keinen durch das NFS erzwungenen, netzwerkweiten, eindeutigen Namen. Aus diesem Grund kann die Ortstransparenz nur durch zusätzliche Maßnahmen gewährleistet werden. Das NFS unterstützt kein dynamisches Migrations-Konzept. Wenn ein Dateisystem dynamisch gemountet wird, muss manuell in jedem Client die Mount-Tabelle aktualisiert werden. Die Konsistenz der Daten kann vom NFS nicht garantiert werden (z.B. neue Dateien

können 30 sek. existieren, ohne dass andere Clients dies erfahren). Das NFS unterstützt kein Replikationsmanagement und die Skalierbarkeit des Systems ist limitiert (Caching-Verfahren wurde nur für kleine lokale Netze entworfen).

3.2 Andrew File System

Das Andrew File System (AFS) wird seit 1983 im Rahmen von Andrew Project [Borenstein (1996)] an der Carnegie Mellon University (CMU), Pittsburgh, PA in Kooperation mit IBM (weiter-) entwickelt. In [Nydick u. a. (2001)] ist detailliert beschrieben, warum mit der Entwicklung eines neuen verteilten Dateisystems begonnen wurde, obwohl das NFS schon zur Verfügung stand und warum das NFS nicht ausreichend war. Das Hauptziel der AFS - Entwickler war die Skalierbarkeit des Systems. Das AFS wurde für den Einsatz auf dem CMU Campus mit ca. 10000. Rechnern entwickelt.

Das AFS besitzt eine Ansammlung von „vertrauenswürdigen“ Servern, alle anderen gelten als unsicher. Die Server sind zustandsbehaftet, d.h. Clients werden z.B. vom Server benachrichtigt, wenn sich Daten, auf die sie gerade zugreifen, geändert wurden. Jeder Server kennt eine komplette Liste von (Haupt-) Verzeichnissen und deren Fundorten. Das AFS erreicht eine hohe Effizienz und Skalierbarkeit durch ein gezieltes Caching auf den Client-Rechnern. Es werden immer ganze Dateien auf die Client-Seite kopiert und dort im lokalen Dateisystem gecachet. Erst wenn die Datei geschlossen wird, erfolgt die Aktualisierung der Server-Seite. Die Datei bleibt weiterhin auf der Client-Seite im Working-Set, bis wieder auf diese zugegriffen wird oder sie aus dem Working-Set verdrängt wird.

In dem Projekt „The dice project. A Comparison Between AFS and NFSv4“ [DICE] an der University of Edinburgh werden das Andrew File System mit dem Network File System v4 verglichen.

3.3 BitTorrent

BitTorrent ist ein von Bram Cohen entwickeltes File-Sharing Protokoll [Cohen (2003)], das unter Einsatz der File-Distribution-Technik für Tauschbörsen gedacht ist.

Die Grundarchitektur von BitTorrent basiert auf dem Hybrid-P2P-Netzwerk, wobei der Server nicht im traditionellen Sinne arbeitet. BitTorrent hat keinen zentralen Server. Stattdessen finden sich die einzelnen Benutzer über einen so genannten Tracker zum Downloaden einer Datei zusammen und laden untereinander hoch. Der Tracker teilt jedem Peer (Client) die Adressen aller anderen Peers mit und sammelt gleichzeitig Informationen darüber, welche Peers welche Datei-Teile besitzen. In der Praxis bekommt ein Peer vom Tracker eine

Zufallsliste mit den Peers, mit denen er Verbindung aufbauen kann. Der Tracker koordiniert die Up- und Download-Aktionen aller Peers. Die Abbildung 3.2 veranschaulicht die BitTorrent Architektur.

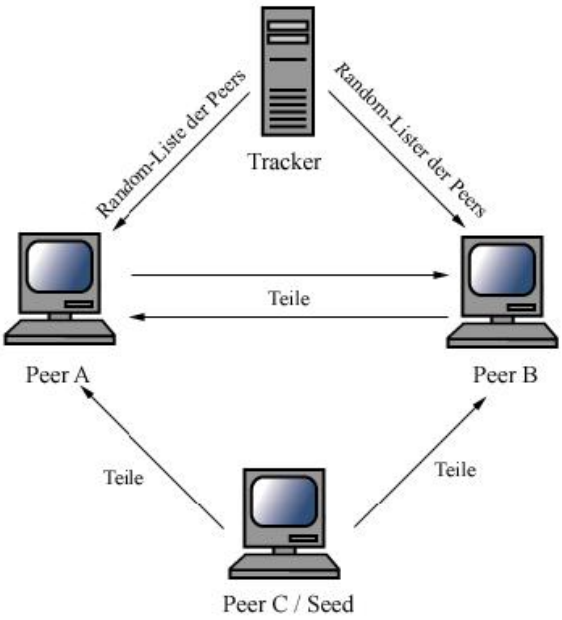

Abbildung 3.2: BitTorrent Architektur

Die Peers unterscheidet man in zwei Kategorien:

- **Seed** ist ein Peer, der die vollständige Datei besitzt und diese anderen Peers zum Download bereitstellt. Ein solcher Peer ist entweder der ursprüngliche Anbieter dieser Datei oder der, der nach einem kompletten Download einer Datei diese weiter zum Download zur Verfügung stellt.
- **Leecher** ist ein Peer, der die noch nicht komplette Datei heruntergeladen hat.

In neuen Versionen wurden „trackerlose“ Systeme entwickelt. Die Trackerfunktion wird dabei von den Clients mit übernommen. Dadurch wird u.a. die bisher fehlende Ausfallsicherheit des Trackers vermieden. Der Einsatz von einem Tracker ist weiterhin möglich. Neue Systeme erleichtern auch das Anbieten der Dateien, da der Tracker den aufwendigsten Teil in dem BitTorrent System ist. In diesem Fall kann der Tracker dezentral, als Distributed Hash Table [Cates (2003)] auf den Clients (im Netz) selbst abgelegt und verwaltet werden. Distributed Hash Table kann den Tracker vollständig ersetzen. Bei einem „trackerlosen“ BitTorrent-System gewinnt man an Fehlertoleranz, da ein Tracker ein Single Point of Failure ist. Ein solches System könnte als ein verteiltes Dateisystem eingesetzt werden.

Eine der wichtigsten Komponenten des BitTorrent Systems ist die *.torrent*-Datei. Solche Dateien sind sehr klein (ungefähr zehn kByte), aber enthalten alle wichtige Metainformationen über die eigentlichen Nutzdaten (Dateien), die heruntergeladen werden sollen. Die Metadaten beinhalten folgende Informationen: Dateiname, Dateigröße, die Hashinformation und die Adresse des Track-Servers.

Clients können das BitTorrent System zu jedem beliebigen Zeitpunkt betreten und verlassen, ohne das das Gesamtsystem gefährdet wird. Dies ist ein sehr großer Vorteil, wenn man an den BitTorrent Einsatz in einem kollaborativen Raum denkt. Ein weiterer Vorteil kommt aus den Peer2Peer Netzwerken, dass es keine feste Bindung zwischen einem Objekt und seinem Speicherort gibt. Es existieren keine zentralen Einheiten und keine Hierarchie unter den Knoten (Clients).

Als Nachteil muss man berücksichtigen, dass keine Schreiboperation in dem System vorgesehen ist. Wenn ein Client eine Datei zum Download bereit stellt, wird sie aus Sicht der anderen Clients read-only. Nur nach dem vollständigen Download der Datei können andere Client sie verändern. Dadurch entsteht natürlich eine neue Datei. Das Löschen eines Objektes wird auch nur durch seinem Besitzer verwaltet, andere Clients haben keinen Einfluss darauf. Wenn ein Client sicher stellen will, dass ein Objekt zu einem späteren Zeitpunkt noch verfügbar ist, muss er es herunterladen und lokal speichern. BitTorrent stellt leider keine Suchfunktionalität zur Verfügung.

BitTorrent ist derzeit eine der besten Möglichkeiten, große Mengen legaler Daten schnell zu verbreiten, und ist sehr gut skaliert.

3.4 Google File System

Das Google File System (GFS) [Gobioff u. a. (2003)] wurde von Google Labs entworfen, um die schnell wachsende Anfrage von Google's Datenverarbeitungsprozessen zu verbessern. Das GFS ist ein Vertreter der Cluster-Filesystem mit folgenden Zielen:

- hohe Performanz
- Skalierbarkeit
- Zuverlässigkeit
- Verfügbarkeit

Als weitere Motivation der Google-Entwickler war die Idee, dass die Fehler-Toleranz und das Auto-Recovery in das System integriert werden sollen.

Ein GFS Cluster besteht aus einem Master und mehreren Chunkservern und ist für eine Vielzahl von Clients zugreifbar. Jeder dieser Server ist in der Regel ein Linux Server. Der strukturelle Aufbau eines Clusters ist in der Abbildung 3.3 dargestellt.

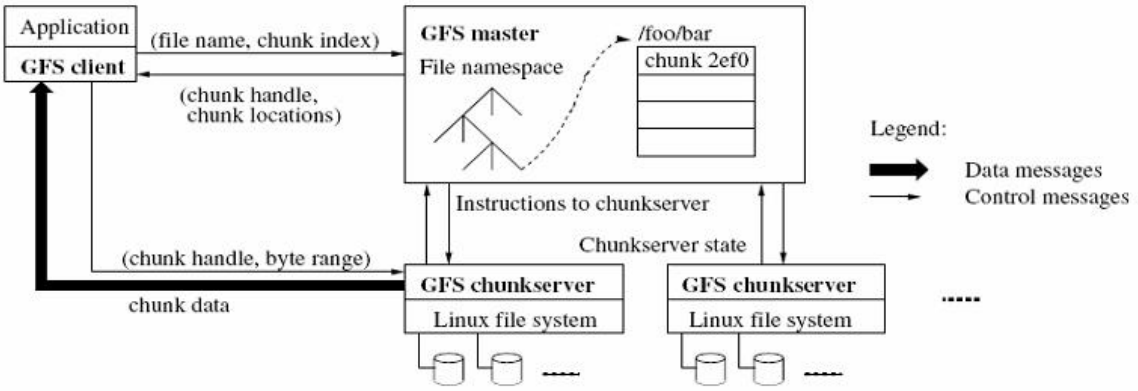

Abbildung 3.3: Google File System Architektur [Gobioff u. a. (2003), S.3]

Die Dateien werden in Chunks fester Größe unterteilt. Jeder Chunk ist mit einem unveränderlichen, globalen und einzigartigen 64 bit chunk handle identifiziert, der beim Erzeugen von dem GFS-Master zugewiesen wird. Chunkserver speichern diese auf die lokale Platte als Linux-Dateien. Für die Zuverlässigkeit wird jeder Chunk auf multiple Chunkserver repliziert. Als default sind drei Kopien von den GFS-Entwicklern vorgesehen. Lesend und schreibend werden Clients nie über den Master zugreifen, sondern ein Client fragt den Master, mit welchem Chunkserver er sich verbinden soll. Weiterhin werden Systemaktivitäten wie Chunk-Lease-Management, Garbage Collection und Chunk Migration zwischen den Chunkservern vom GFS-Master koordiniert.

Für das Google File System, als ein Beispiel für ein Cluster-Filesystem, sind der Umgang mit riesengroßen Datenmengen und die intelligente Ausbreitung der Arbeitslast sehr bedeutsame Aspekte. Dies wird dadurch erreicht, dass die eigentlichen Nutzdaten von den Metadaten getrennt gespeichert und verwaltet werden. Bei solcher Trennung können mehrere Speicherorte eingesetzt werden, ohne die Integrität des Gesamtsystems zu gefährden. Der GFS-Master hält alle Metadaten des Dateisystems. Diese beinhalten Datei- und Chunk-Namensräume, Zugriffsrechte sowie das Mapping von Dateien nach Chunks und die Position jeder Chunkkopie.

Da der GFS-Master nur koordiniert, hat er ein geringeres Datenvolumen als das Gesamtsystem zu handhaben. Die Replikation des GFS-Masters, als eine Redundanz der zentralen Komponente bei einem möglichen Ausfall, ist dadurch auch möglich.

Jede Mutation verändert die Chunkdaten und wird auf allen Chunk-Kopien durchgeführt. Leases (Kontrollrechte) werden hierbei benutzt, um eine konsistente Mutationsreihenfolge zu garantieren. Der GFS-Master wählt eine primäre Kopie aus, und gibt ihr das chunk lease, also das Recht der Wahl der Mutationsreihenfolge auf allen untergeordneten Chunk-Kopien. Die globale Mutationsreihenfolge ist somit durch die Leaserechte des Masters sowie durch die von der primären Kopie festgelegten Mutationsreihenfolge unter den anderen Kopien definiert.

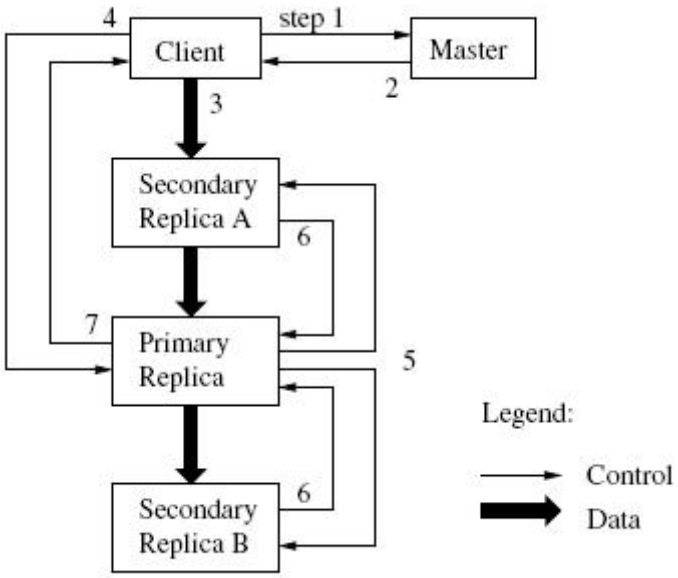

Abbildung 3.4: Schreibkontrolle und Datenfluss in GFS [Gobioff u. a. (2003), S.5]

Die Abbildung 3.4 stellt den Ablauf einer Schreiboperation in dem Google File System dar. Der Ablauf der Schreiboperation ist in [Gobioff u. a. (2003)] beschrieben.

3.5 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 Abbildung 3.5 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

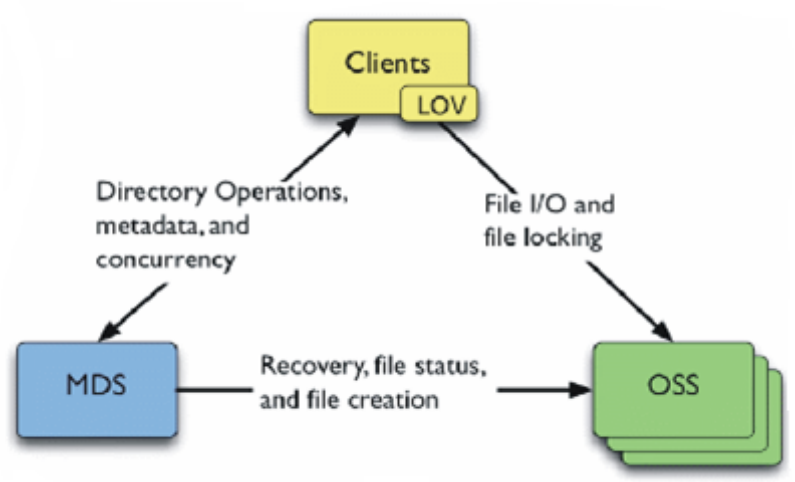

Abbildung 3.5: Interaktion zwischen Systemen in Lustre File System

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.

4 Zusammenfassung

Mit dieser Ausarbeitung möchte der Autor einige verteilte Dateisysteme vorstellen. Die dabei gewonnenen Erkenntnisse sind für die Auswahl eines verteilten Systems notwendig, das im Rahmen vom „Collaborative Workspace“ Projekt [Kabalkin (2007a)] genauer untersucht wird. Werden die Untersuchungen zeigen, dass ein ausgewähltes verteiltes Dateisystem den definierten Anforderungen entspricht, wird der Autor das System im Rahmen seiner Master Thesis einsetzen. Die ersten Ideen über die Master Thesis sind in [Kabalkin (2007b)] beschrieben.

Die Tabelle 4.1 stellt den Zusammenhang der ersten Untersuchungen der in Kapitel 3 vorgestellten verteilten Dateisysteme dar.

VERTEILTES DATEISYSTEM	NFS	AFS	BITTORRENT	GOOGLE FS	LUSTRE FS
Zugriffstransparenz	+	+	-	+	+
Ortsunabhängigkeit	+/-	+	+	+	+
Skalierbarkeit	+/-	+	+/-	+	+
Redundanz	-	+	-	+	+
Konsistenz von Daten	-	+	-	+	+
Fehlertoleranz und Sicherheit	+/-	+	+/-	+	+
Hardware und Betriebssystem Heterogenität	+	+	+	?	+/-

Tabelle 4.1: Vergleich verteilter Dateisysteme.

Ein wichtiger Aspekt, den bei der Auswahl von einem Distributed File System berücksichtigt werden sollte, sind die Kosten. Der Autor entschied sich für den Einsatz von einem Open Source Distributed File System. Leider steht das Google File System nicht zur Verfügung. Das System entspräche allen Anforderungen, wurde aber vermutlich nur für interne Abläufe von Google Inc. entwickelt.

In dem „Collaborative Workspace“ Projekt wird sich der Autor dieser Ausarbeitung mit dem Lustre File System auseinandersetzen und das System detaillierter untersuchen.

Literaturverzeichnis

[DICE] : *A Comparison Between AFS and NFSv4*. The dice project, University of Edinburgh. – URL http://www.dice.inf.ed.ac.uk/groups/services/file_service/docs/newfs-choice.html

[NFSv4] : *General Information and References for the NFSv4 protocol*. – URL http://www.nfsv4.org/

[OpenAFS] : *OpenAFS*. – URL http://www.openafs.org/

[Borenstein 1996] BORENSTEIN, N. S.: *CMU's Andrew project: a retrospective*. 1996

[Cates 2003] CATES, Josh: *Robust and Efficient Data Management for a Distributed Hash Table*, Massachusetts Institute of Technology, Diplomarbeit, Juni 2003. – URL http://pdos.csail.mit.edu/papers/chord:cates-meng.pdf

[Chang u. a. 2006] CHANG, Fay ; DEAN, Jeffrey ; GHEMAWAT, Sanjay ; HSIEH, Wilson C. ; WALLACH, Deborah A. ; BURROWS, Mike ; CHANDRA, Tushar ; FIKES, Andrew ; GRUBER, Robert E.: *Bigtable: A Distributed Storage System for Structured Data*. OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA. November 2006. – URL http://labs.google.com/papers/bigtable-osdi06.pdf

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

[Cluster File Systems, Inc. 2006] Cluster File Systems, Inc. (Veranst.): *Lustre 1.4.7 Operations Manual*. Version 1.4.7.1-man-v36 (11/30/2006). November 2006. – URL https://mail.clusterfs.com/wikis/lustre/LustreDocumentation?action=AttachFile&do=get&target=LustreManual36.pdf

[Cohen 2003] COHEN, Bram: *Incentives Build Robustness in BitTorrent*. Mai 2003. – URL http://www.bittorrent.com/bittorrentecon.pdf

[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

[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: *Persistenz-Service System*. Februar 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master06-07/kabalkin/abstract.pdf

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

[Noveck u. a. 2003] NOVECK, D. ; SHEPLER, S. ; CALLAGHAN, B. ; ROBINSON, D. ; THURLOW, R. ; BEAME, C. ; EISLER, M.: *Network File System (NFS) version 4 Protocol*. April 2003. – URL http://tools.ietf.org/html/rfc3530

[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