UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Verteilte Dateisysteme in der Cloud

Cloud Data Management

Seminararbeit

Leipzig, Januar 2010

vorgelegt von

Maria Moritz

Dozent: Prof. Dr. Erhard Rahm
Abteilung: Datenbanken

Inhaltsverzeichnis

1 Einleitung

1.1 Cloudcomputing

Unternehmen wie Google, Yahoo oder Amazon betreiben riesige, zweckspeziefische Architekturen, um ihre Anwendungen zu unterstützen und Services im Bereich Verarbeitung oder Speicherung anzubieten. Cloud-Computing übernimmt nun die Rolle, zum Zwecke von Kostenvorteilen Services, Datenverarbeitung oder Daten selbst separat auf einer zentralisierten Anlage oder bei einem Betreiber anzubieten. Dadurch kann es einfacher und oft billiger sein, auf Daten zuzugreifen oder deren Wert durch bessere Zusammenarbeit oder Analysezwecke auf einer gemeinsamen, einheitlichen Platform zu steigern [Cre09]. Cloud-Computing kann in drei Bereiche aufgeteilt werden:

- **SaaS (software-as-a-service):** Aplikations-Servieces wie Google Apps, Salesforce.com oder WebEx
- **PaaS (platform-as-a-service):** grundlegende Elemente zur Entwicklung neuer Anwendungen (z.B. Coghead, Google Application Engine)
- **IaaS (infrastructure-as-a-service):** bietet Rechen-und Speicher-Infrastruktur als zentralisierten Dienst an (Amazon)

Dabei sind verteilte Dateisysteme mit ihrer Aufgabe, Daten zügig und dauerhaft bereit zu stellen, bei Punkt drei (IaaS) anzusiedeln. Die folgende Arbeit wird einige bekannte verteilte Dateisysteme vorstellen und auf Implementierungsaspekte eingehen.

1.2 Anforderungen an verteilte Dateisysteme

Verteilte Dateisysteme organisieren und verarbeiten Massen an Daten, die von Anwendungen für wissenschaftliche, betriebliche oder private Zwecke verarbeitet werden und somit einer breiten Masse von Benutzern zugänglich gemacht werden. Sie bilden die Grundlage für den Datenaustausch von Cloud-Anwendungen und anderen großen, über das Internet zur Verfügung gestellten Diensten. Dabei werden Terabytes an Daten auf Tausenden Maschinen verteilt und gleichzeitig von Hunderten Benutzern angefragt. Um diesen Aufgaben gerecht zu werden, müssen Anforderungen an die Implementierung solcher Dateisysteme gestellt werden, die sowohl Datenverfügbarkeit und Dauerhaftigkeit, als auch eine schnelle Bereitstellung angeforderter Daten umfasst. Dabei sollten sie bei hoher Fehlertoleranz auch auf handelsüblichen Maschinen laufen. Die folgenden Kapitel sollen einige bekannte verteilte Dateisysteme vorstellen und darauf eingehen, wie diese den oben genannten Anforderungen begegnen.

1.3 Gliederung

Die übrigen Kapitel sind wie folgt aufgebaut: Kapitel 2 beschreibt das Google File System. Es beschreibt wie Google mit dem rapide anwachsenden Anspruch von Datenverarbeitungs-Anforderungen umgeht, und wie es Datenkonsistenz bei gleichzeitig hoher Verfügbarkeit unterstützt. Kapitel 3 befasst sich mit Hadoop Distributed File System (HDFS), einem höchst fehlertoleranten, verteilten Dateisystem, das entwickelt wurde, um auf handelsüblicher Hardware zu laufen. Kapitel 4 stellt Amazon S3 vor, ein Speicherservice der im Rahmen der Amazon Web Services angeboten wird. Hier wird darauf eingegangen, inwieweit S3 das Arbeiten mit Forschungsnetzen wie z. B. DZero unterstützt. Da sich dieses Kapitel vor Allem auf das Paper [PIRG08] stützt, unterscheidet es sich im inhaltlichen Aufbau von den vorangegangen. Zuletzt wird eine kurze Zusammenfassung über die beobachteten Eigenschaften der verschiedenen Dateisysteme gegeben.

2 GoogleFS

Google FS (Google File System) wurde entwickelt, um den wachsenden Anforderungen der Datenverarbeitung von Google zu begegnen. Dabei teilt es viele Ziele mit vorangegangenen verteilten Dateisystemen, wie z. B. Performanz und Sklaierbarkeit, möchte dabei aber auch Problemen wie Anwendungs- und Plattenfehlern begegnen und eine konstante Fehlerentdeckung, sowie Monitoring integrieren.

2.1 Aufbau im Überblick

Ein GFS-Cluster besteht aus einem einzelnen Master und mehreren Chunkservern. Wobei es sich um gewöhnliche Linux-Maschinen handelt, auf denen Server-Prozesse laufen. Dateien werden in einzelne Chunks gesplittet und auf den Chunkservern lokal gespeichert. Identifiziert werden sie über sogenannte Chunk-Handles, die vom Master zur Erstellungszeit der Chunks bestimmt werden. Zum Zwecke der Zuverlässigkeit werden Replikationen jedes Chunks auf mehreren Chunkservern gespeichert. Siehe Abbildung 1.

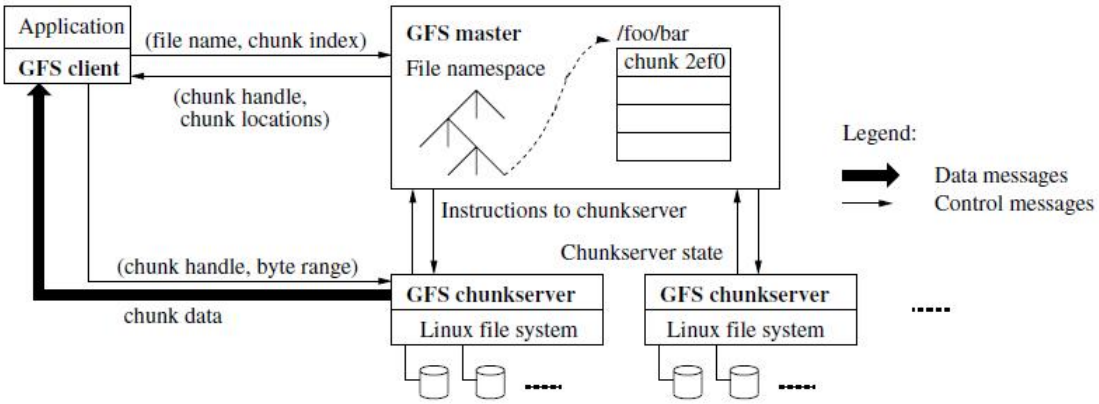

Abbildung 1: Architektur des Google FS [GGL03]

Der Master organisiert alle Dateisystem-Metadaten. Er verwaltet den Namensraum, Informationen zur Zugangskontrolle und dem aktuellen Aufenthalt der Chunks und übernimmt das Mapping von Dateien auf Chunks. Der Master kommuniziert mit den Chunkservern über sogenannte Heartbeat-Messages, um ihnen Instruktionen zu übermitteln und deren Status abzufragen. Clients kommunizieren mit dem Master und den Chunkservern, um Daten im Auftrag der Anwendungen lesen oder schreiben zu können. Dabei interagiert der Client mit dem Master, um Metadaten-Operationen auszuführen, der Datenfluss hingegen läuft direkt über die Chunkserver.

Single Master

Clients fragen beim Master nach, welche Chunkserver sie kontaktieren müssen, um Dateien erhalten, bzw. schreiben zu können. Die Informationen über die Chunkserver werden für mehrere aufeinander folgende Operationen gecached. Hierbei sendet der Client dem Master in einem Request den Dateinamen und den Chunkindex des entsprechenden Chunks. Der Master Antwortet mit dem Chunk-Handle und dem Ort der Replikation. Der Client cached diese Informationen, indem er den Dateinamen und den Chunkindex als Schlüssel benutzt. Nun wird ein Request an den nächst liegenden Chunkserver gesendet. Solange die Informationen über den Chunk im Cache liegen, können Anfragen an den Chunkserver gestellt werden, ohne wieder mit dem Master kommunizieren zu müssen. Das minimiert einen Flaschenhals, zu dem der Master werden könnte.

Konsistenzmodel

Änderungen an Metadaten werden ausschließlich vom Master ausgeführt und in seinem Operation-Log gespeichert. Der Zustand einer Dateiumgebung hängt davon ab, ob die Änderung gelingt oder fehl schlägt oder ob es konkurrierende Änderungen gab. Eine Dateiumgebung ist konsistent, wenn alle Clients die selben Daten sehen, egal von welcher Replikation gelesen wird. Eine Umgebung ist definiert, wenn sie konsistent ist und die Clients sehen, was geändert wurde. Konkurrierende aber erfolgreiche Veränderungen machen die Umgebung konsistent aber undefiniert: Alle Clients sehen die gleichen Daten, aber nicht, welche Veränderung was geschrieben hat. Eine gescheiterte Änderung macht die Gegend unkonsistent, also auch undefiniert.

GFS wendet die Änderungen eines Chunks in der selben Reihenfolge auf alle Replikationen an und nutz Chunk-Versionsnummerierung um ungültige Replikationen zu finden, die aufgrund eines Serverausfalls keine Updates erfahren haben. Um Datenverlust durch Komponentenausfälle zu vermeiden, kommuniziert der Master mit den Chunkservern über regelmäßige „Heartbeats“. Dabei benutz er Prüfsummen, um beschädigte Daten zu finden. In diesen Fällen werden die Daten schnellstmöglich durch Replikationen ersetzt.

Sämtliche Google-Applikationen ändern Dateien eher durch das Hinzufügen von neuen Daten, als durch Überschreiben. Das ist effizienter und dynamischer gegenüber Anwendungsfehlern als wahlfreies Überschreiben. Durch regelmäßige Checkpoints, die den Status des Masters repräsentieren, kann die Start-Up-Zeit des Masters gering gehalten werden. Außerdem erlauben sie zyklisches Schreiben und vermeiden das Lesen von unvollständigen Daten aus Anwendungsperspektive.

2.2 Systemprozesse

Leases

Um eine konsistente Reihenfolge von Veränderungen von Replikationen zu organisieren, werden sogenannte Leases genutzt. Der Master wählte eine Chunkreplikation aus, die Primary genannt wird. Dieser legt die Reihenfolge für die Änderungen des Chunks fest. Alle anderen Replikationen folgen dieser Ordnung. Dieser Lease-Mechanismus ist entworfen worden, um den Organisationsoverhead des Masters zu minimieren. Solange der Chunk verändert wird, kann der Primary mit dem Master via Heartbeat-Messages kommunizieren.

Datenfluss

Der Datenfluss wurde vom Kontrollfluss entkoppelt. Während Steuerbefehle vom Client, üder den Primary zu allen anderen Replikationen führen, werden Daten linear, entlang einer Kette von Chunkservern geleitet. Dabei ist es Ziel, die Netzwerkbandbreite jedes Rechners auszunutzen und Latenzzeiten zu minimieren. Außerdem werden die Daten über TCP transportiert und sofort weiter geleitet. Da vollduplex Links benutzt werden, wird das Empfangen von Daten nicht vermindert.

Schnappschüsse

Die Schnappschussoperation wird verwendet, um Kopien von Datei- oder Verzeichnisbäumen zu erstellen oder den aktuellen Status des Verzeichnisses zu sichern, um ihn nach Änderungen leichter zu committen oder rückgängig zu machen. Wenn der Master eine Schnappschussanfrage erhält, entzieht er zunächst allen Chunks die ausstehenden Leases. Das stellt sicher, dass folgende Schreiboperationen auf diese Chunks Interaktion mit dem Master erfordern, um den aktuellen Inhaber des Leases zu finden, was wiederum dem Master die Gelegenheit gibt, zunächst eine neue Kopie des Chunks zu erstellen. Danach beginnt der Master die Metadaten für die Quelldateinen bzw. -verzeichnisse zu kopieren. Die neu erstellten Schnappschussdateien zeigen auf die gleichen Chunks wie die Quelldateien.

2.3 Masteroperationen

Namensraumorganisation und Locking

Neben der Ausführung aller Namensraumoperationen übernimmt der Master auch die Organisation über das Replizieren von Chunks und deren Platzierung. Außerdem wird die Auslastung

der Chunkserver kontrolliert.

Da Masteroperationen im Allgemeinen sehr lange dauern, nebenläufige Operationen aber nicht verzögert werden sollen, werden Locks auf bestimmten Bereichen des Namensraumes benutzt. GFS stellt seinen Namensraum lokal als Lookup-Tabelle, in der die Pfadnamen auf die Metadaten gemappt werden, dar. Jede Masteroperation nimmt eine Reihe von Locks vor, bevor sie ausgeführt wird. Zum Beispiel wird auf die Pfade /d1, /d1/d2, ..., /d1/d2/.../dn eine Lesesperre gesetzt und auf /d1/d2/.../dn/leaf eine Schreib- oder Lesesperre, wenn letzteres in eine Operantion einbezogen wird. Eine Lesesperre auf den Elternverzeichnissen ist ausreichend, um das Löschen dieser zu verhindern, weil kein Verzeichnis oder Datenstruktur vor Veränderungen geschütz werden muss. Durch dieses Locking ist es möglich konkurrierende Änderungen auf verschiedenen Dateien im gleichen Verzeichnis vorzunehmen.

Garbage-Collection

Genau wie alle anderen Änderungen loggt der Master das Löschen einer Datei. Doch anstatt den Speicherplatz sofort zurück zu nehmen, wird die Datei mit einem versteckten Namen, der den Löschzeitpunkt enthält, versehen. Beim regelmäßigen Scannen des Dateisystems werden die Dateien gelöscht, die älter als beispielsweise drei Tage sind. Bis dahin kann die Datei gelesen und wieder hergestellt werden. Beim endgültigen Löschen werden die Links auf die zugehörigen Chunks gelöscht und in einem weitern Scan diese „verwaisten“ Chunks.

Replikationsverwaltung

Sobald der Master eine Replikation erstellt, muss entschieden werden, wo diese platziert werden soll, um Speicher optimal ausnutzen zu können und sie über mehrere Racks zu verteilen. Der Master re-repliziert einen Chunk soblad die Anzahl der Replikate unter einen bestimmten Wert fällt, z.B durch Chunk-Serverausfall oder durch benutzerspeziefische Erhöhung der Replikationszahl. Dabei wird nach verschiedenen Kriterien beurteilt, welcher Chunk zunächst re-repliziert wird. Beispielsweise werden aktive Dateien eher repliziert, als bereits gelöschte Objekte.

2.4 Verfügbarkeit

Eine der größten Herausforderungen beim Entwurf eines verteilten Dateisystems ist es, den häufig auftretenden Fehlern und Ausfällen von Komponenten zu begegnen. Im Folgenden sollen die Maßnahmen und Tools erläutert werden, die diese Probleme vermeiden, bzw. beheben können.

Fast Recovery

Sowohl der Master als auch der Chunk-Server sind so aufgebaut, dass sie ihren Status speichern können und so innerhalb weniger Sekunden neu starten können, egal wie sie beendet wurden. Dabei wird nicht zwischen normalem Beenden oder fehlerhaftem Abbruch unterschieden, die Prozesse der Server werden routinemäßig durch Killbefehl beendet.

Chunk Replikation

Wie bereits erwähnt, werden Chunks auf veschiedenen Servern und Racks repliziert. Dabei kann der Benutzer die Replikations-Level auf den verschiedenen Bereichen des Namensraumes selbst bestimmen. Der Standard ist drei. Der Master übernimmt hierbei das Re-replizieren soblad Chunkserver offline gehen oder beschädigte Replikation melden.

Master Replikation

Der Operations-Log des Masters wird auf mehreren Maschinen repliziert, um dessen Verfügbarkeit zu gewährleisten. Erst wenn ein Zustand geloggt ist und lokal, wie auch auf allen Replikationen des Masters gespeichert wurde, wird eine Veränderung zugelassen.

Ein Master-Prozess hat hierbei die Verantwortung für diese Veränderungen und für Hintergrund-Aktivitäten, wie Garbage-Collection. Wenn dieser ausfällt, kann er sofort neu starten. Wenn dessen Platte oder Rechner ausfällt, startet eine Monitoring-Infrastructur, die außerhlab von GFS liegt, auf einer anderen Maschine und mit dem replizierten Operations-Log einen neuen Master-Prozess. [GGL03]

3 HadoopFS

Hadoop Distributed File System (im Weiteren HDFS) ist ein höchst fehlertolerantes, verteiltes Dateisystem, das entwickelt wurde, um auf handelsüblicher Hardware zu laufen. Es ist geeignet für Anwendungen mit großen Datensätzen, bietet einen schnellen Zugriff auf diese Daten und wurde ursprünglich für Apache-Nutch, einer Open-Source-Suchmaschine entwickelt.[Bor07]

3.1 NameNodes und DataNodes

HDFS vertritt eine Master-Slave-Architektur. Jedes HDFS-Cluster besteht zum Einen aus einem NameNode, also dem Master, der den Namensraum des gesamten Dateisystems organisiert und den Zugang zu den Dateien durch die Clients regelt, sowie den sogenannten DataNodes, die das Speichern von Daten der ihnen anhaftenden Knoten verwalten.

Der NameNode führt Operationen auf dem Namensraum des Dateisystems aus, wie z. B. das Öffnen, Schließen und Umbenennen von Dateien und Verzeichnissen. Außerdem sind sie für das Mapping von Datenblöcken auf mehrere DataNodes verantwortlich. Die DataNodes wiederum bedienen die Lese- und Schreiboperationen der Clients und führen das Blockerstellen, -löschen und Replikationen im Auftrag der NameNodes durch. Siehe Abbildung 2

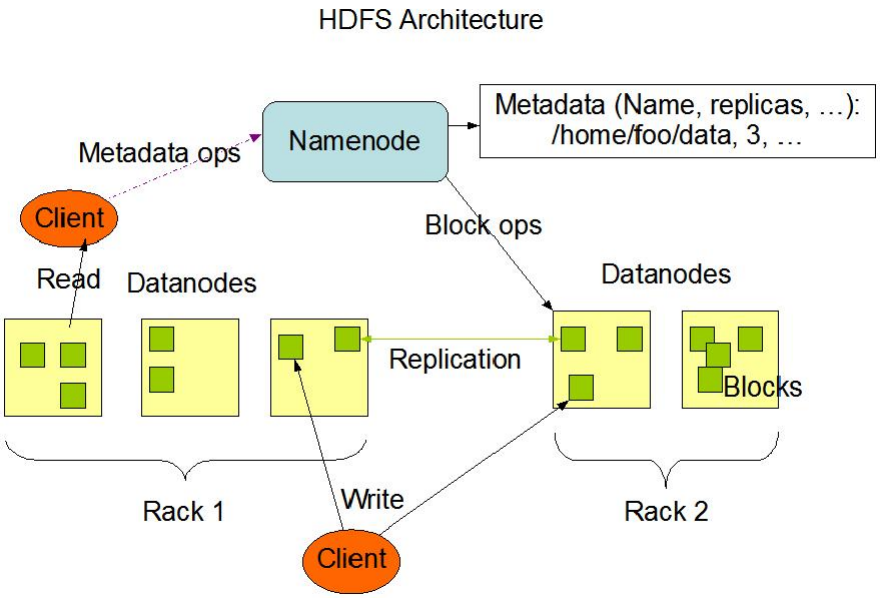

Abbildung 2: Architektur des HadoopFS [Bor07]

HDFS ist eien Java-Implementierung, daher kann jede java-unterstützende Maschine die NameNode oder DataNode-Software ausführen. Typischerweise läuft in einer konkreten Konfigurati-

on die NameNode-Software auf einem einzelnen Rechner und die DataNode-Software ebenfalls auf jeweils einem Rechner des Clusters. Es ist von Seiten der Architektur jedoch nicht unmöglich auch mehrere DataNode-Instanzen auf einer Maschine auzuführen, tritt in der Praxis jedoch seltener auf.

3.2 Datenreplikation

Das HDFS soll es ermöglichen, sehr große Dateien über mehrere Maschinen verteilt, in einem weiten Cluster, zuverlässig zu speichern. Dateien werden als Block-Sequenz abgespeichert. Diese Blöcke werden repliziert, um Fehletoleranz zu gewährleisten. Dabei kann die Applikation die Anzahl der Replikationen einer Datei selbst bestimmen. Dateien werden in HDFS einmalig geschrieben und besitzen auch nur einen Schreiber zur Zeit.

Die Entscheidungen über die Replikationen von Dateien liegt beim NameNode. Er bekommt periodisch sogenannte „Heartbeat-Messages“ und Blockrepots, die eine Liste aller Blöcke des DataNodes beinhalten. Somit ist Kenntnis über das einwandfreie Funktionieren des Knotens zugesichert.

Replikationsersetzung

Größere HDFS-Instanzen laufen auf einem Computer-Cluster, das über mehrere Racks verteilt ist. Die Kommunikation zwischen zwei Knoten in verachiedenen Racks muss durch Switches geleitet werden. Meistens ist die Netzwerkbandbreite zwischen zwei Maschinen des gleichen Racks höher als die zwischen Maschinen in unterschiedlichen Racks.

Der NameNode bestimmt die Rack-ID, die jeder DataNode enthält. Eine einfache, jedoch nicht optimale Lösung wäre es, Replikationen in einheitlichen Racks zu speichern, um Datenverlust zu vermeiden, falls ein ganzes Rack ausfällt. Außerdem ermöglicht diese Verteilung der Replikation eine einfache Lastbalance. Jedoch werden durch diesen Ansatz die Kosten für Schreibprozesse deutlich erhöht, da Blöcke über mehrere Racks transportiert werden müssen.

Im allgemeinen Fall wird der Replikationsfaktor drei gewählt. Davon liegen zwei Replikationen auf unterschiedlichen Knoten des selben Racks, eine dritte auf einem DataNode eines separaten Racks. Somit wird der Schreibverkehr zwischen den Racks reduziert, und dennoch die Datenverfügbarkeit und -Beständigkeit garantiert, da ein Rack-Ausfall wesentlich seltener vorkommt, als ein Knotenausfall.

Replikationsauswahl

Um Bandbreite zu sparen, versucht HDFS einen Lese-Request zu bedienen, indem die dem Leser am nächsten liegende Replikation genutzt wird. Wenn eine Replikation im selben Rack existiert, wird diese bevorzugt. Spannt sich ein HDFS-Cluster über mehrere Datencenter, wird die Replikation des lokalen Datencenters bevorzugt.

3.3 Persistenz der Metadaten

Der Namensraum von HDFS wird von den NameNodes gespeichert. Diese benutzen einen Transaktionslog, das EditLog, um Änderungen auf den Metadaten, wie z. B. das Erzeugen einer neuen Datei oder das Ändern des Replikationsfaktors einer Datei, dauerhaft zu speichern. Dieses Log wird gemeinsam mit einer sogenannten FsImage-Datei, die den gesamten Namensraum des Dateisystems, sowie das Mapping der Dateiblöcke auf die Dateien beinhaltet, lokal beim NameNode gespeichert.

Beim Starten eines NameNodes wird das FsImage und das EditLog von der Platte gelesen, alle Transaktionen des EditLogs zum geladenen FsImage hinzu geführt und diese neue Version des FsImages auf die Platte ausgegeben. Dieser Vorgang wird als Checkpoint bezeichnet und soll nach künftigen Implementationen periodisch auftreten.

Daten des HDFS werden in lokalen Dateien der DataNodes gespeichert. Dabei werden Datenblöcke über Dateien und Vetzeichnisse verteilt gespeichert. Beim Starten eines DataNodes wird das lokale Dateisystem durchgeschaut und eine Liste aller Datenblöcke erstellt, die dann als Blockreport an den NameNode gesendet wird.

3.4 Robustheit

Die häufigsten Probleme, die in einem verteilten Dateisystem auftreten können, sind NameNode-Asufälle, DataNode-Ausfälle und Netzwerk-Partition.

DataNode-Ausfälle

Durch den regelmäßigen Heartbeat, den ein DataNode an den NameNode sendet, kann schnell festgestellt werden, ob ein oder mehrere Knoten aufgrund von Netzwerkpartition ausgefallen sind. Diese Knoten werden dann als tot markiert, an sie werden keine weiteren I/O-Requests mehr gesendet. Durch den Ausfall von DataNodes kann die Replikationszahl eines Blocks unter den festgelegten Wert fallen. Der NameNode findet die neu zu replizierenden Blöcke und initiiert Replikationen, wann immer es nötig wird.

Datenintegrität

Wenn Datenblöcke von einem DataNode gelesen werden, kann es vorkommen, dass diese wegen Netzwerkfehlern oder fehlerhaften Speichermedien beschädigt sind. Deshalb implementiert die HDFS-Client-Software Prüfsummen-Kontrolle. Beim Erstellen einer Datei durch einen Client, wird eine Prüfsumme zu jedem Block der Datei ermittelt und in einer versteckten, separaten Datei des selben HDFS-Namensraums gespeichert. Beim Empfangen von Dateiinhalten verifiziert der Client die Daten, die er von den verschiedenen DataNodes erhalten hat, mithilfe der Prüfsummen. Falls Fehler auftreten, kann der Client den entsprechenden Block von einem anderen DataNode, der eine Replikation enthält, anfragen.

NameNode-Ausfälle

Da das FsImage und das EditLog zentrale Datenstrukturen des HDFS sind, muss die Unversehrtheit dieser gesichert werden. Zu diesem Zweck kann der NameNode mehrere Kopien des FsImage und des EditLog verwalten. Aufgrund von Updates auf den Datenstrukturen, sowie den Kopien, wird die Zahl der Transaktionen pro Sekunde auf dem Namensraum reduziert. Das ist jedoch vertretbar, da HDFS-Applikationen zwar datenintensiv, jedoch nicht Metadatenintensiv sind. [Bor07]

4 Amazon S3

Amazon Web Services, eine hundertprozentige Tochter von Amazon.com, stellt seit 2007 auch in Deutschland ein Speicher-Hilfsmittel zur Verfügung, genannt Simple Storage Service (S3). Im folgenden Kapitel soll dieser vorgestellt werden und kurz darauf eingegangen werden, inwieweit Amazon S3 die Arbeit mit Forschungsnetzwerken wie DZero [1] unterstützten könnte.

4.1 Architekturüberblick

S3 wird seit seiner Einführung von einer breiten Masse Benutzer, Privatpersonen wie auch Unternehmen verwendet und bietet niedriege Datenzugriffszeit, sowie sehr gute Dauerhaftigkeit und Verfügbarkeit. S3 speichert Daten über zwei Ebenen. In sogenannten Buckets (ähnlich Ordnern) können User ihre Daten speichern, außerdem wird hierüber die Gebührenerhebung organisiert. In diesen Buckets wiederum können unbegrenzt viele Datenobjekte gespeichert werden, die aus einem Namen, einem Daten-BLOB und Metadaten bestehen. Benutzer können Objekte lesen, schreiben oder verändern, wobei das Umbenenn oder Verschieben eines Objektes, das Herunterladen und neu Hochladen desselben unter einem neuen Namen erfordert. Gebühren werden für das Speichern (pro GB und pro Monat), das Transferieren (Upload und Download) und für Operationen (get, put, list) erhoben. Hierbei werden alle Operationen dem Besitzer des Buckets in Rechnung gestellt, nicht dem des Objektes.

Benutzer registrieren sich bei Amazon Web Services mit einem Public- und einem Private-Key. Jedes S3-Account ist für Zahlungszwecke mit einer Kreditkarten-Nummer verbunden. Zugangskontrolle wird auf Bucket- bzw. Objekt-Ebene umgesetzt. Dabei können Buckets konfiguriert werden, um Zugangs-Log-Einträge mit Informationen zu Objekt, Zeit und Request-Typ zu speichern.

4.2 Amazon S3 Evaluation

Die folgende Evaluierung [PIRG08] wird durch die Anforderungen von Forschungsnetzen wie DZero an Amazon S3 bestimmt. DZero ist ein Gemeinschaftsprojekt, das sich mit dem grundlegenden Wesen der Materie beschäftigt und im Fermi National Accelerator Laboratory (Fermilab) in Batavia, Illinois, USA durchgeführt wird.

Zunächst wurden fünf PlanetLab-Knoten [2] die in ihrer Positionierung der von DZero entsprechen, benutzt. Zwei befanden sich in Europa (einer in Deutschland und einer in Frankreich), zwei in den USA (New York und Kalifornien), der fünfte Knoten repräsentierte eine ausgewählte Maschine der Universität von Süd-Florida. Außerdem wurden weitere Messreihen mit Hilfe

[1] The DZero Experiment. http://www-d0.fnal.gov
[2] PlanetLab Consortium. http://planet-lab.org

von Servern innerhalb der Amazon EC2 Cloud erstellt, um die Performance von S3 frei von Störungen des Internets zu messen.

Dauerhaftigkeit

Aufgrund begrenzen Budgets wurde die Dateigröße der Experimente von minimal 1KB bis auf 1GB beschränkt. Es wurden Tests mit 10.000 Dateien während einer Testperiode von zwölf Monaten und 137.000 Anfragen von und zu S3-Servern durchgeführt. Es trat während dieser Zeit kein Fall von anhaltendem Datenverlust auf.

Datenverfügbarkeit

Es wurde ein S3-Bucket mit Testobjekten verschiedener Größe (1KB, 1MB, 16MB, 100MB) angelegt. Während des Testlaufes wurden diese Daten zufällig geschrieben und die vorgespeicherten Daten gelesen. Zwischen den Tests wurden Verzögerungen eingefügt. Im Fehlerfall wurden die Anfragen bis zu fünfmal wiederholt, dann wurde ein Fehler deklariert. Von 107.556 Tests auf EC2 ergaben sich fünf Instanzen bei denen eine HTTP PUT-Anfrage aufgrund eines Fehlers wiederholt werden musste und 23 bei denen der Request ohne Antwort-Code (hier keine weiteren Wiederholungen) ablief. Nur fünf der getesteten Lesezugriffe erforderten Wiederholungen.

Während eines Versuches über die Verfügbarkeit von S3 wurde ein und dasselbe Objekt über einem Zeitraum von 23 Wochen in einem Intervall von 15 Minuten (in der Summe 15.456 Anfragen) neu herunter geladen. Dabei wurde eine Verfügbarkeit von 95,89% nach dem originalen Download, 98,06% nach der ersten Wiederholung, 99,01% nach der zweiten, 99,76% nach der dritten, 99,94% nach der vierten Wiederholung und 100% Verfügbarkeit nach der fünften Wiederholung gemessen. Diese Tests wurden dabei sowohl durch S3's internes System beeinflusst, als auch durch die Internet-Konnektivität zwischen Amazon und der Universität von Süd-Florida.

Datenzugriffsperformanz

Im folgenden Kapitel soll die Datenzugriffszeit auf S3-gespeicherte Daten bewertet werden. Dazu wird die Download-Geschwindigkeit von Dateien der Größen 1B, 1KB, 1MB, 16MB und 100MB mit einer Gesamtzahl von 13.178 Zugriffen gemessen.

Abbildung 3 zeigt die Download-Geschwindigkeiten der verschiedenen Objektgrößen. Daraus geht hervor, dass für kleine Objekte der Transaktions-Overhead einen großen Teil der Zugriffszeit aus macht und S3 also für die Speicherung größerer Objekte geeigneter ist. Außerdem wird deutlich, dass maximal 100 Objekte pro Sekunde herunter geladen werden können. Die maxi-

male Bandbreite liegt bei etwa 21 MB pro Sekunde. Alle Tests wurden für Uploads wiederholt und entsprachen in etwa denen der Downloads.

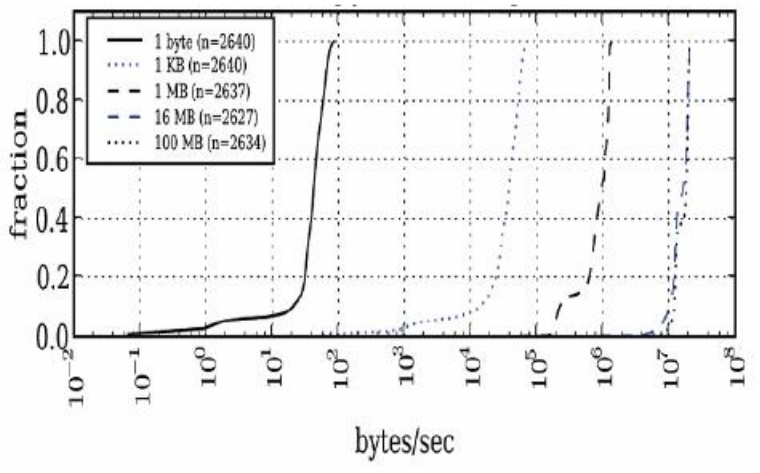

Abbildung 3: Zugriffszeiten auf Objekte der Größe 1B, 1KB, 1MB, 16MB und 100MB [PIRG08]

Konkurrente Performanz In einem weiteren Versuch wurde der wiederholte Zugriff von zwei virtuellen Maschinen verschiedener EC2-Cluster (usma1 und usma2) auf Daten desselben Buckets getestet. Während einer Dauer von 11 Stunden wurden pro VM jeweils für zehn Minuten ein Thread, dann zwei usw. für GET- und PUT-Operationen von 100MB gestartet. Abbildung 4 und Abbildung 5 zeigen, dass mit Zunahme der Threads zwar die Bandbreite pro Thread abnimmt, dafür aber die Gesamt-Bandbreite zunimmt. So wurde eine Gesamtbandbreite von ca. 30 MB pro Sekunde erreicht.

Remote Access Performance Um Zugriffszeiten von verschiedenen Standorten zu messen, wurden 28 Experimente über 7 Tage (vier pro Tag) durchgeführt. Abbildung 6 zeigt, dass der Durchsatz vom Standort abhängig ist.

4.3 Amazon S3 for Science Grids

Im folgenden Abschnitt werden Eigenschaften bezüglich Science-Grids diskutiert und wie diese mit Hilfe von DZero-Experimenten getestet wurden.

Kosten

Es müssem zwei Arten von Kosten unterschiden werden: Datenzugriffskosten und Datenspeicherkosten. Für ein Forschungsnetz wie DZero wären das pro Jahr $691.200 Speicherkosten

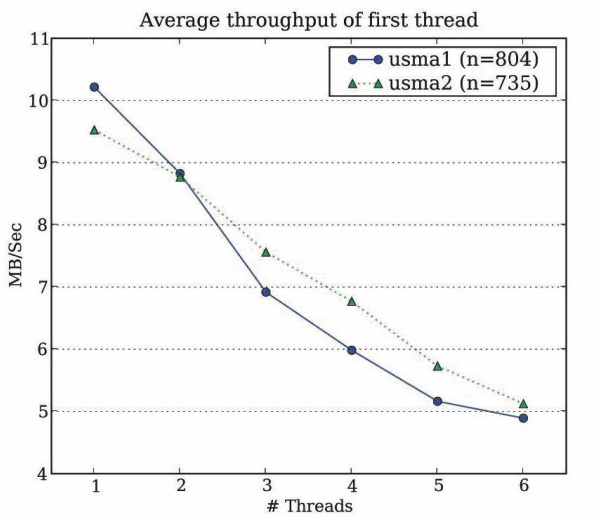

Abbildung 4: Durchschnittlicher Durchsatz des ersten Threads [PIRG08]

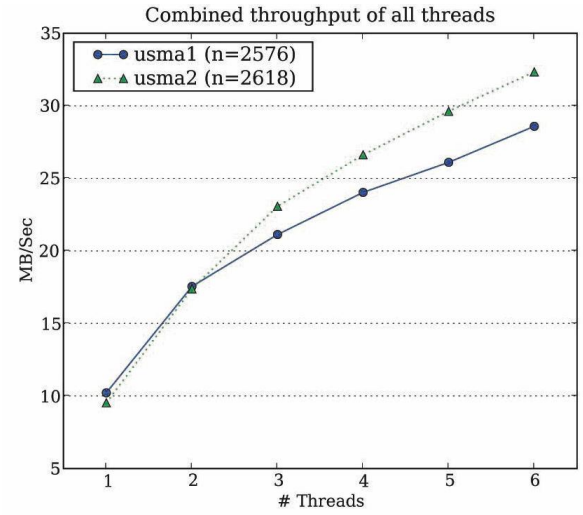

Abbildung 5: Durchschnittlicher Durchsatz aller Threads [PIRG08]

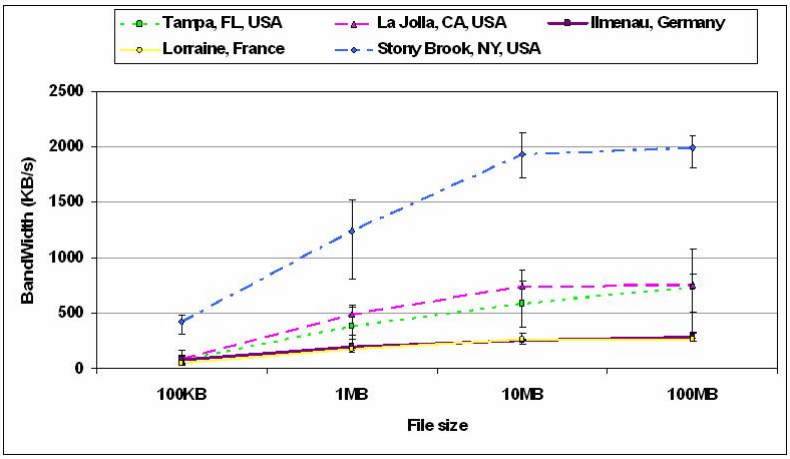

Abbildung 6: Download-Bandbreite verschiedener Standorte und Dateigrößen [PIRG08]

und $335.012 Datentransferkosten. Im Folgenden sollen verschiedene Ansätze zum Reduzieren beider Kostentypen vorgestellt werden.

Da ein signifikanter Anteil von DZero-Daten nur für eine begrenzte Zeit benötigt wird, z.B werden über 30% der Daten nicht länger als einen Tag bentutzt, ist es nicht notwendig diese in hochverfügbarem Speicher zu halten. Stattdessen können diese Daten in langsamerem, weniger teurem Speicher archiviert werden. Ebenfalls zu erwähnen ist die Möglichkeit, nur Originaldaten zu speichern und weitere Daten von diesen abzuleiten.

Transferkosten könnten durch lokale Caches verringert werden. Experimente mit veschiedenen Cache-Ersetzungsalgorithmen haben gezeigt, dass lokale Caches in TB-Größe den Datentransfer von permanenten Speichermedien stark reduzieren kann, da die Größe der Daten, die pro Auftrag transferiert werden oft 2,5% der Cachegröße nicht übersteigt. Außerdem können Transferkosten verringert werden, indem man die Verarbeitung nahe an den Speicherort der Daten bringt. So werden die Übertragungen zwischen S3 und Amazon's Elastic Computer Cloud (EC2) nicht angerechnet. Oder Transferkosten werden durch Kosten zur Berechnung abgeleiteter Daten ersetzt, was wesentlich billiger ist. Obwohl alle oben genannten Ansätze durch datenintensives Computing bestätigt wurden, benötigen sie Unterstützung von Anwendungsseite oder von S3.

Performanz

Nun soll die Frage geklärt werden, ob Outsourcing der Datenspeicherung auf Kosten der Performanz geht.

Offensichtlich betreibt S3 verschiedene Platzierungsstrategien, abhängig von dem Benutzer, der ein Bucket erstellt. Das erklärt die Unterschiede im Downloading 6 und erfordert außerdem das

Ändern der Datenplatzierung, um Performanzunterschiede zu minimieren.

Für Stapelverarbeitung mit wenig Benutzerinteraktion beeinflusst der relativ langsame Zugang zu indivduellen Daten, den S3 anbietet, die Performanz nicht sonderlich, solange S3 die Daten schneller zur Verfügung stellt, als sie weiter verarbeitet werden können. Daher wird ein System, das S3-gehostete Daten benutz, nur limitierten Speicher benötigen und ein gut konzipiertes Job-Management, das Daten holt, solange alte Jobs noch in der Computerqueue liegen. [PIRG08]

5 Zusammenfassung

Um große Anwendungen oder Dienste für betriebliche, private oder wissenschaftliche Zwecke betreiben zu können ist es nötig Infrastrukturen zu schaffen, um die Verfügbarkeit der zugehörigen Daten zu sichern und trotzdem Skalierbarkeit zu berücksichitgen. Cloudcomputing ermöglicht es dabei, Hardware optimal ausnutzen zu können, Kosten zu sparen oder Datenwert durch gemeinschaftliche Nutzung zu steigern.

Verteilte Dateisysteme ermöglichen es, Benutzern eine Vielzahl von Diensten oder Anwendungen zur Verfügung zu stellen. Dabei ist es für die Betreiber maßgebend den Anforderungen an große Anwendungen zu begegnen. Die unten stehende Tabelle fasst die Umsetzung einzelner Dateisysteme nocheinmal kurz zusammen. Dazu ist noch zu sagen, dass über Amazon S3 keine Aussagen zur Implementierung gemacht werden und daher auf Dynamo einem Key-Value-Store von Amzon referenziert wurde [DHJ+07].

	GoogleFS	**HadoopFS**	**Amazon S3**
Architektur	> Cluster mit Master und Chunkservern, > Master identifiziert Chunks über Chunk-Handles, > Clients erfragen Daten beim Master	> NameNodes verwalten Namensraum, > DataNodes verwalten Daten und Replikationen im Auftrag der NameNodes	> Benutzer speichern ihre Daten in Objekten, > diese widerum sind in Buckets organisiert
Verfügbarkeit/ Persistenz	> Operation-Log auf allen Chunk-Replikationen, > Hearbeats um Chunkstatus zu übermitteln	> Replikationen auf Data-Nodes, > sequentielles, einmaliges Schreiben, > Blockreports	> Replikationen auf mehreren Datencentern (Dynamo) > asynchrone Updates auf Replikationen (Dynamo)
Performanz	> Data-Appendings statt Overwrites, > Kontrollfluss über Clients Datenfluss über Chunks	> Replikationsauswahl aus lokalem Cluster	> verschiedene Platzierungsstrategien, abhänging vom Ort des Benutzers
Robustheit/ Sicherheit	> Handshakes, um > Komponentenausfälle zu vermeiden > Replikationen auf Chunks, > Replikation des Master-Operation-Log	> FSImage, EditLog, > Heartbeats & Blockreports	> siehe Verfügbarkeit > Registrierung mit Public und Private-Key > Zuganskontrolle auf Bucket- und Objektebene

Literatur

[Bor07] Dhruba Borthakur. HDFS Architecture. The Apache Software Foundation, 2007. http://hadoop.apache.org/common/docs/r0.20.1/hdfs_design.pdf.

[Cre09] Mache Creeger. Cloud computing: An overview. *Queue*, 2009. http://portal.acm.org/citation.cfm?id=1554608.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazons Highly Available Key-value Store. In *SOSP*, 2007. http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003. http://labs.google.com/papers/gfs-sosp2003.pdf.

[PIRG08] Mayur Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. Amazon S3 for Science Grids: a Viable Solution? In *DADC '08: Proceedings of the 2008 international workshop on Data-aware distributed computing*, 2008. http://portal.acm.org/citation.cfm?id=1383526.

Abbildungsverzeichnis