



Google File System

Seminar WS 2007-2008

Student: Tien Duc Dinh

Betreuer: Olga Mordvinova, Julian Kunkel

Datum: 04-12-2007

[Gliederung]

Inhalt

- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- 3. HDFS & Demo

1. Einleitung
 - Motivation
 - Grundbegriffe
2. GFS im Überblick
 - Architektur
 - Datenstruktur
 - Operationen
 - Systemeigenschaften
3. Hadoop Distributed File System und Demo

[Motivation]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

■ Google File System (GFS)

- o ein verteiltes Dateisystem von Google
- o für die Internetsuche entwickelt
- o multi-GB Dateien effizient verwaltet
- o für Systeme aus günstiger Hardware, die häufig ausfällt
- o ermöglicht nebenläufigen Zugriff von mehreren Rechnern auf eine Datei
- o hat folgende spezielle Eigenschaften:
 - Monitoring
 - Fehlertoleranz
 - automatische Zustandwiederherstellung (wenn Fehler auftreten)

[Grundbegriffe (1)]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

■ Master

- o einzelner Prozess in einer separaten Maschine
- o verwaltet Dateisystem-Metadaten
- o verwaltet System-Aktivitäten

[Grundbegriffe (2)]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

■ Metadaten

- o Daten, die Informationen über andere Daten enthalten
- o z.B. Chunk-Namensraum, Chunk-Locations

■ Mutation

- o ist eine Operation, die den Inhalt von Metadaten eines Chunks verändert
- o z.B. write, append Operationen

[Grundbegriffe (3)]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

■ Chunk

- o ähnlich wie ein Block
- o 64 MB groß

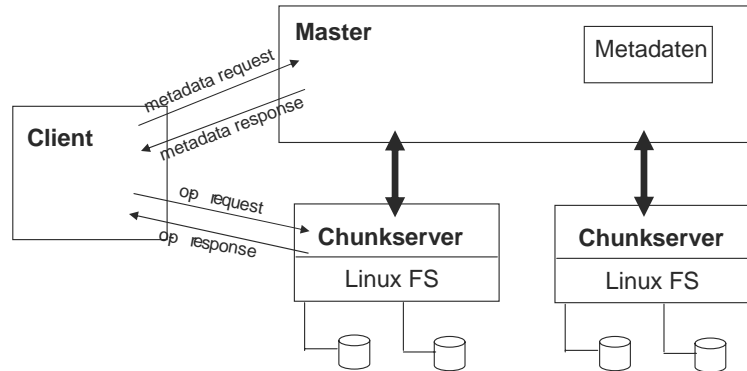
■ Chunkserver

- o eine Schnittstelle, die Chunks auf lokalen Festplatten als Dateien verwaltet

GFS im Überblick

Inhalt

- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- 3. HDFS & Demo

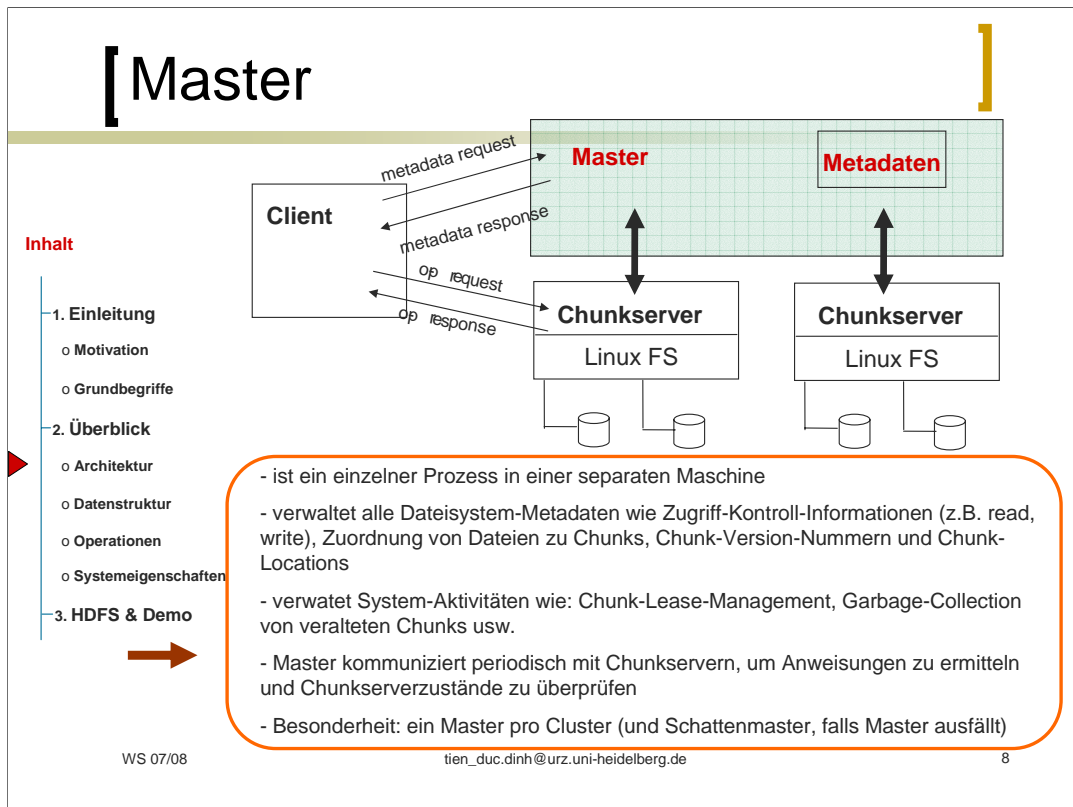


WS 07/08

tien_duc.dinh@urz.uni-heidelberg.de

7

- Ein GFS Cluster besteht aus einem Einzelmaster, mehreren Chunkservern und wird von mehreren Clients zugegriffen. Dateien werden in Chunks in fester Größe aufgeteilt.



Master

- Mit nur einem Einzelmaster ist das Design viel einfacher und es ermöglicht, komplizierte Anweisungen wie Chunkplatzierung (eng. chunk placement) und Chunk-Replication zu implementieren. Das Problem dafür ist, Master muss so wenig wie möglich beteiligt sein, damit er nicht zum Flaschenhals kommt und deshalb führen Clients Operationen wie read und write nie auf Master aus, sondern fragt Master, welchen Chunkserver Clients verbinden sollen, dann speichern Clients die Informationen für eine begrenzte Zeit und kommunizieren mit Chunkservern.

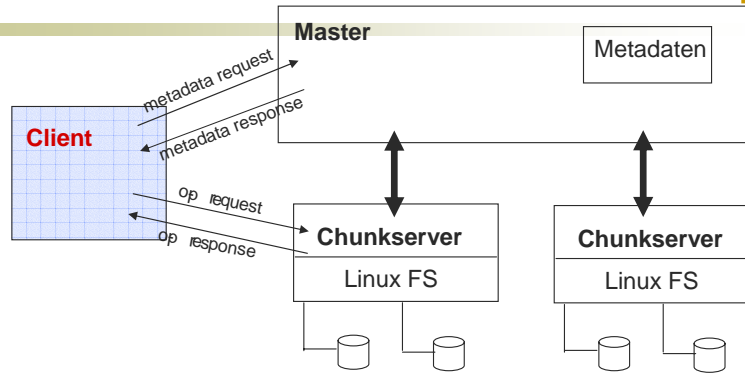
Metadaten

- Alle Metadaten werden im Masterspeicher gespeichert
- Master enthält drei Typen von Metadaten: Datei- und Chunk-Namensraum, Zuordnung von Dateien zu Chunks, Chunk-Version-Nummern und Chunk-Locations
 - + Die ersten 3 Typen sind permanent gespeichert. Bei Logging-Mutationen werden sie persistent in Log-Dateien auf Master gespeichert. Die Log-Dateien sind nützlich, um den Masterzustand einfach und zuverlässig wiederherzustellen, falls er abstürzt
 - + Der 4. Typ ist nicht permanent im Masterspeicher gespeichert, sondern Master fragt Chunkserver nach Chunk-Locations beim Master-Neustarten und wenn ein Chunkserver den Cluster verbindet -> Der Grund dafür ist, man möchte vermeiden, dass Master und Chunkserver synchronisiert sind, z.B. wenn Chunkserver den Cluster verlassen oder verbinden, ausfallen, neustarten usw. In einem Cluster mit hundertern von Chunkservern passiert dieser Vorgang sehr oft.

Client

Inhalt

- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- 3. HDFS & Demo

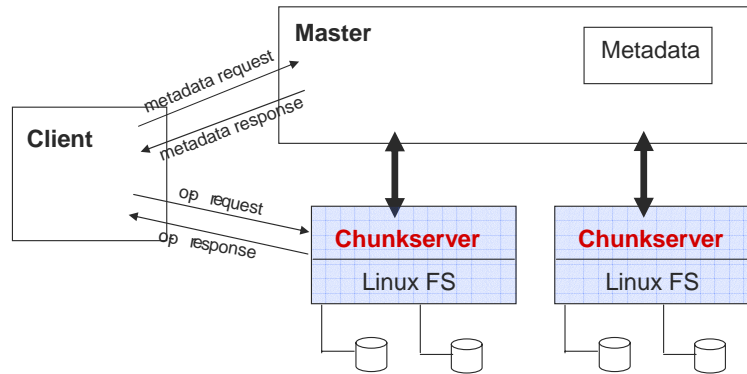


- ist ein GFS API Interface von Applikation
- kommuniziert nur mit Master (wegen Metadaten)
- read/write Operationen werden zwischen Client und Chunkserver direkt ausgeführt → Master wird nicht alleine zum Bottleneck (Flaschenhals)

[Chunkserver]

Inhalt

- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- 3. HDFS & Demo



- ist ein Interface, welches Chunks als Dateien verwaltet
- kommuniziert nur mit Master (wegen Metadaten)
- Operationen werden zwischen Client und Chunkserver direkt ausgeführt

[Chunk (1)]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

- ähnlich wie ein Block
- 64 MB groß
- als Datei in Chunkservern gespeichert
- jeder Chunk 3 mal (konfigurierbar) auf mehreren Chunkservern repliziert. (Chunk-Replicas)
- Chunk-Version
- Chunk-Location
- Chunk-Datei-Name (chunk handle)

- Jeder Chunk ist durch seinen Chunk-Datei-Namen (eng. chunk handle) identifiziert, der bei der Chunkerzeugung von Master erstellt wird. Chunkserver speichern Chunks auf lokale Festplatten als Dateien und führen Operationen z.B. read und write mit dem Chunk-Dateinamen und Datei-Größenbereich (eng. byte range) aus.

[Chunk (2)]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

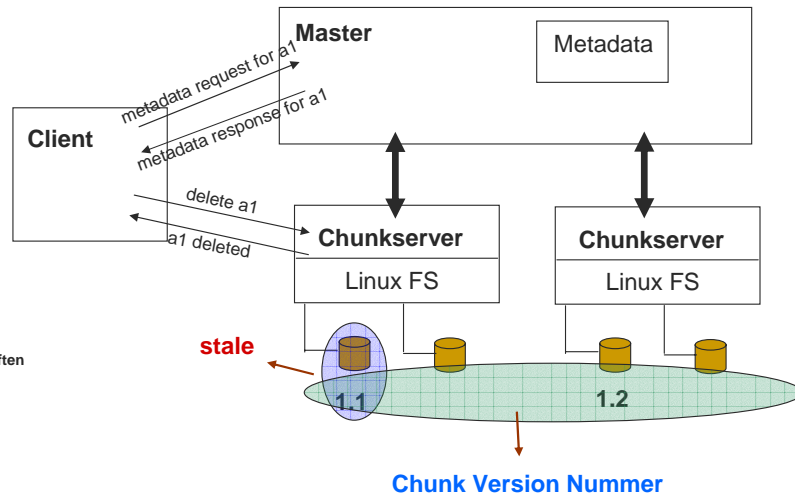
■ Warum sind Chunks gut?

- o wenig Interaktion mit Master
 - weil read und write Operationen von Applikationen meistens auf großen Dateien basieren
- o Metadaten – Größe von Master ist reduziert
- o es vermeidet das Network-Overhead
 - indem eine persistente TCP Verbindung zwischen Client und Chunkserver über eine feste Zeit erstellt wird

Stale Replica Detection

Inhalt

1. Einleitung
 - o Motivation
 - o Grundbegriffe
2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
3. HDFS & Demo



WS 07/08

tien_duc.dinh@urz.uni-heidelberg.de

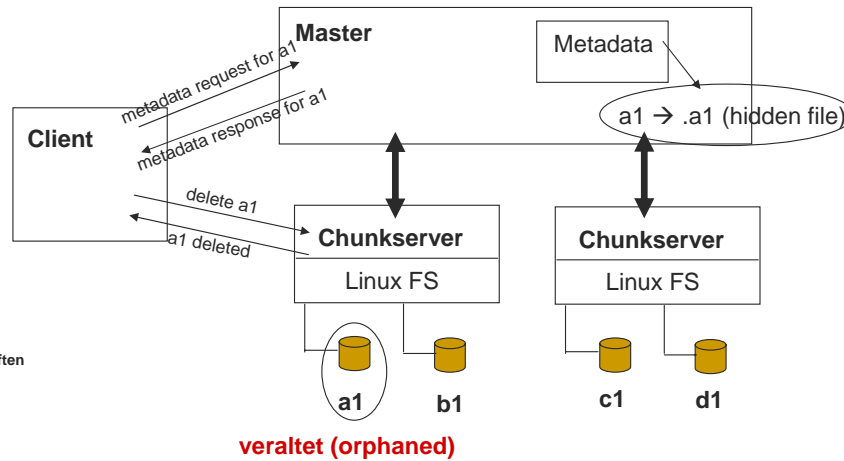
13

- Ein Chunk-Replica kann stale (veraltet) werden, wenn ihr Chunkserver ausfällt und Mutationen auf diesem Chunk verloren hat.
- Master benutzt Chunk-Version-Nummer, um veraltete und aktualisierte Chunk-Replicas zu unterscheiden
- Durch das Scannen von Master werden die veralteten Replicas gelöscht

Garbage Collection von veralteten Chunks (1)

Inhalt

- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- 3. HDFS & Demo



WS 07/08

tien_duc.dinh@urz.uni-heidelberg.de

14

- Wenn eine Datei gelöscht wird, protokolliert Master diese Aktion wie andere Veränderungen. Diese Datei wird zu einer unsichtbaren Datei mit Lösch-Zeittemple (eng. deletion timestamp) umbenannt und ist nur lesbar. Diese Datei kann wieder ganz normal werden, wenn sie zu einer sichtbaren Datei umbenannt wird.

- Nachdem Master den Dateisystem-Namenraum gescannt hat, entfernt er diese Datei. Wenn sie vom Namenraum (nach 3 Tagen, konfigurierbar) entfernt wird, werden ihre Metadaten (z.B. Chunk-Locations, Chunk-Version-Nummern) im Speicher auch gelöscht.

- Durch die Kommunikation zwischen Master und Chunkservern schickt Master Chunkservern einen Bericht mit Chunk-Datei-Namen, welche nicht mehr auf Master-Metadaten sind. Nach einem Vergleich mit dem Bericht löschen Chunkserver die zu löschenden Chunks.

Garbage Collection von veralteten Chunks(2)

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

■ Effizient

- o Garbage Collection läuft im Hintergrund (wie das Scannen vom Namensraum und die Kommunikation zwischen Master und Chunkservern)
- o Kosten sind also amortisiert
- o läuft im Batch
- o wird ausgeführt, wenn Master relativ frei ist. So wird er nicht stark belastet und kann Client-Anfragen rechtzeitig bedienen

■ Zuverlässig

- o durch die Verzögerung des Löschsens von Dateien kann Master auf das zufällige und unumkehrbare Löschen verzichten

[Chunk-Erstellung]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

- Wenn Master ein Chunk erstellt, wählt er dazu einen Ort:
 - o Master wählt Chunkserver mit unterdurchschnittlicher Kapazitätsauslastung
 - o Master limitiert die Anzahl von neuen Chunks auf jedem Chunkserver
 - o Chunk-Replicas werden auf unterschiedliche Racks verteilt

[Chunk Re-replication (1)]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

- Master führt eine Chunk Re-replication durch, wenn eine hohe Anfragemenge vorliegt, und nicht genügend Replicas zur Verfügung stehen.
 - o Chunkserver nicht verfügbar
 - o Replicas korrupt
 - o Diskfehler
 - o Anfragemenge nimmt zu

[Chunk Re-replication (2)]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

■ Bei Re-replication werden Chunks priorisiert:

- o wie weit ist es von der Anfragelast entfernt?
 - z.B. ein Chunk mit einem Verlust von zwei Replicas hat eine höhere Priorität als ein Chunk mit Verlust von einer Replica
- o ein Chunk für gültige Dateien wird gegen über einen für gelöschte Dateien bevorzugt
- o um eine Fehlerauswirkung auf laufende Applikationen zu vermeiden, wird die Priorität von einem Chunk erhöht, welches einen Client-Prozess blockiert hat

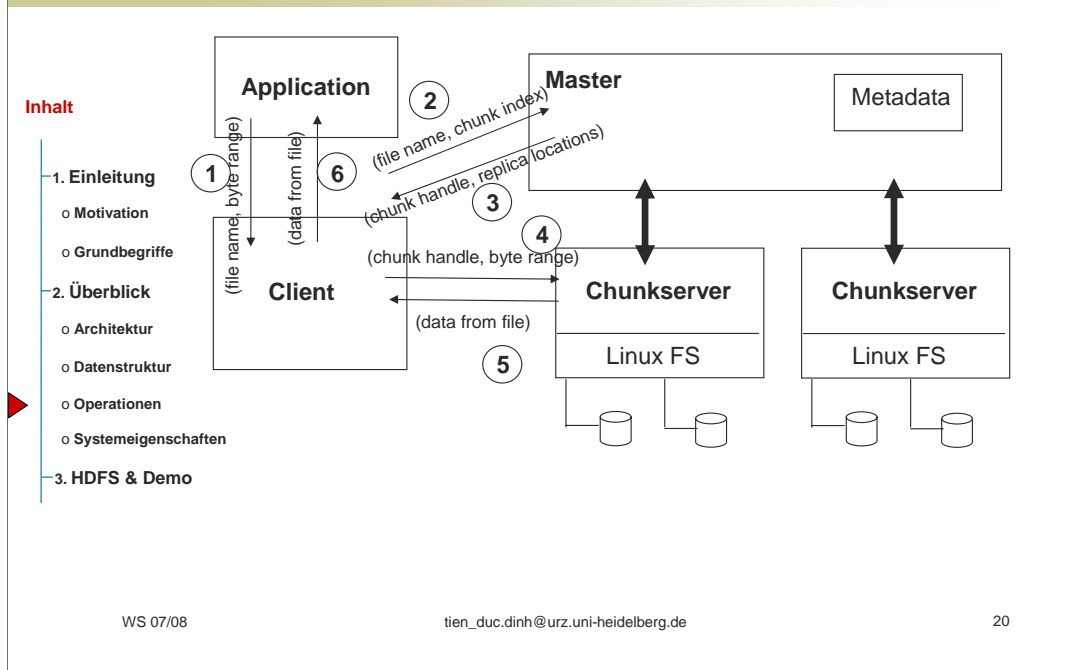
[Chunk Rebalancing]

Inhalt

- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- 3. HDFS & Demo

- Master rebalanciert Replicas periodisch, überprüft die aktuelle Replica-Verteilung und verschiebt Replicas an einen Ort, an dem es keine hohe Last gibt (für eine bessere Diskkapazität und Last-Balancing)
 - o was für ein Replica soll Master verschieben?
 - o → ein Replica bei einer unterdurchschnittlich freien Kapazität

[Read-Algorithmus]



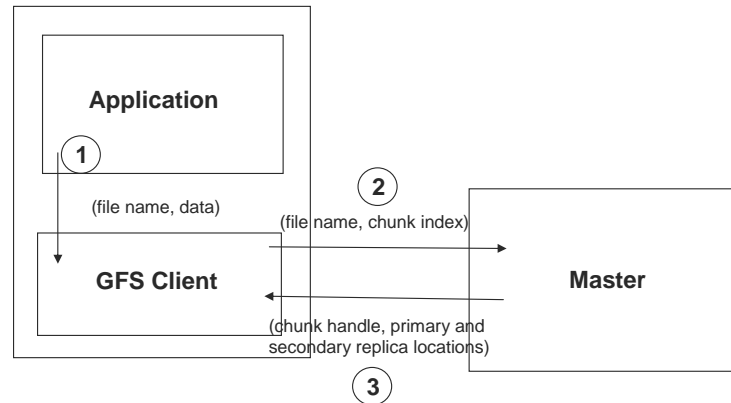
Read-Algorithmus:

1. Applikation erzeugt eine read-Anfrage.
2. GFS Client übersetzt diese Anfrage von (file name, byte range) zu (file name, chunk index) und schickt sie zu Master
3. Master antwortet mit Chunk-Replica-Namen (eng. chunk handle) und Replica-Locations
4. Client wählt eine Location aus (in der Regel die nächste Location) und schickt die Anfrage zu dieser Location.
5. Chunkserver schickt die verlangten Daten zu Client
6. Client schickt die Daten weiter zur Applikation

Write-Algorithmus (1)

Inhalt

- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- 3. HDFS & Demo



WS 07/08

tien_duc.dinh@urz.uni-heidelberg.de

21

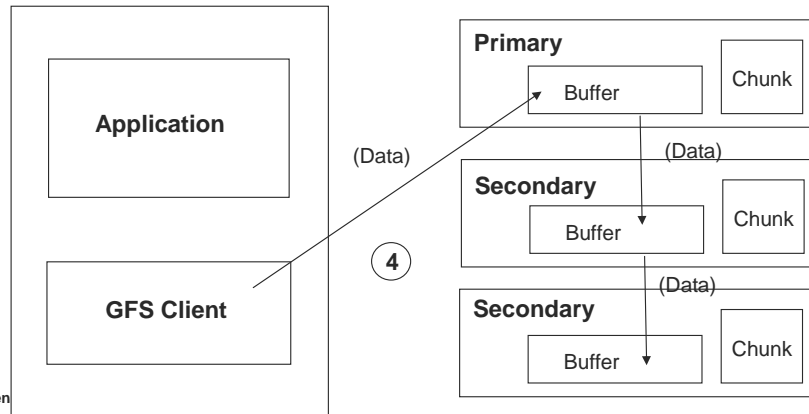
Write-Algorithmus:

1. Applikation erzeugt eine write-Anfrage.
 2. GFS Client übersetzt diese Anfrage von (file name, data) -> (file name, chunk index) und schickt sie zum Master
 3. Master antwortet mit „chunk handle“ und (primary + secondary) Replica-Locations
 4. Client schiebt Daten zu allen Locationen. Daten sind im internen Chunkserverbuffer gespeichert.
 5. Client schickt die write-Operation zum Primary-Replica
 6. Primary-Replica bestimmt serielle Folgen für die im Chunkserverbuffer gespeicherten Dateninstanzen und schreibt diese Instanzen nach dieser Folge auf den Chunk
 7. Primary-Replica schickt die serielle Folgen zu Sekundärreplicas. Sekundärreplicas führen den write-Operation aus.
 8. Secondary-Replicas antworten dem Primary-Replica
 9. Primary-Replica antwortet dann Client
- (Wenn die write-Operation auf Chunkserver ausfällt, wird der Client benachrichtigt und versucht dann diese Operation auszuführen)

Write-Algorithmus (2)

Inhalt

- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- 3. HDFS & Demo



WS 07/08

tien_duc.dinh@urz.uni-heidelberg.de

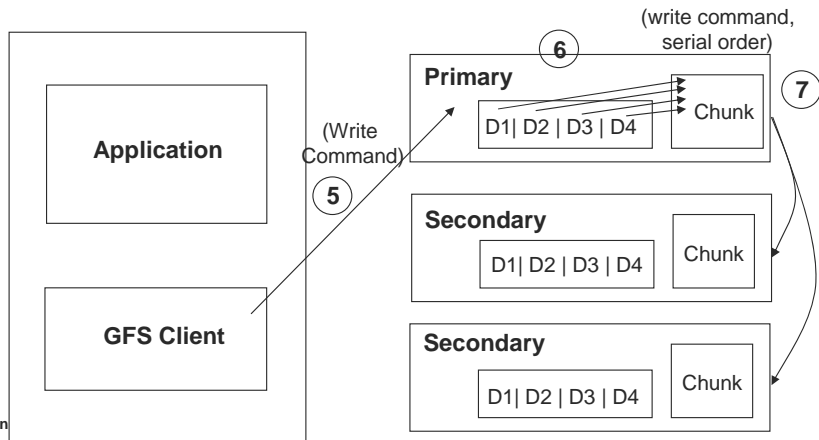
22

Auf dem Bild ist nur ein Beispiel, wie die Daten übertragen werden können. Die Reihenfolge beim Übertragen der Daten ist nicht festgelegt. Client wählt den nächsten Ort, egal ob es Primary oder Secondary ist, und schiebt dann die Daten durch (pipelined). So ist das ganze Netzwerk nicht belastet.

Write-Algorithmus (3)

Inhalt

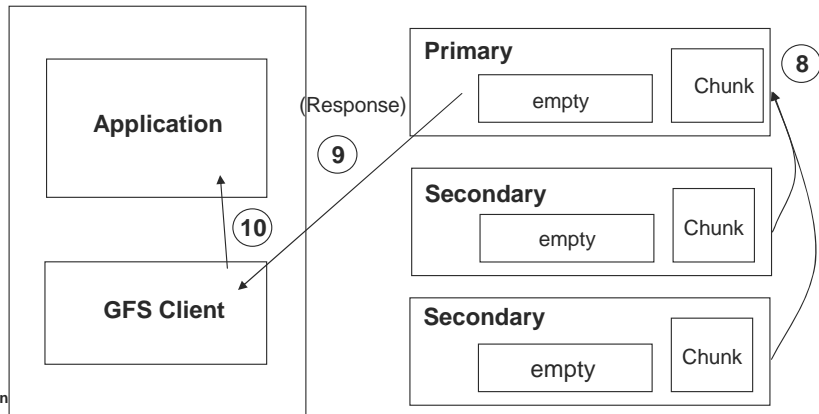
- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- 3. HDFS & Demo



[Write-Algorithmus (4)]

Inhalt

- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- 3. HDFS & Demo



[Record Append (1)]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

■ wichtige Operation bei Google

- o Ergebnisse von mehreren Rechner in eine Datei hinzufügen
- o Datei als eine „producer – consumer“ Queue

Algorithmus

1. Applikation erzeugt eine „record append“ Anfrage
2. GFS Client übersetzt diese Anfrage und schickt sie zum Master
3. Master antwort mit „chunk handle“ und (primary+ secondary) Replica-Locations
4. Client schiebt zu schreibende Daten zu allen Locations
5. Primarychunk überprüft, ob die hinzukommenden Daten an den spezifizierten Chunk angepasst sind

[Record Append (2)]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

6. Wenn nicht angepasst, dann:

- Primarychunk erzeugt einen neuen Chunk
- informiert Secondary-Chunks, diese Aktion auszuführen
- informiert den Client
- Client versucht dann das Hinzufügen auf dem erstellten Chunk

7. Wenn angepasst, dann:

- Primarychunk fügt die Daten in den Chunk hinzu
- informiert Secondary-Chunks, diese Aktion auszuführen
- bekommt Antworten von den Secondary-Chunks
- und schickt die endgültige Antwort zu Client

[Systemeigenschaften]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

■ Fehlertoleranz

- o Chunkreplication: über mehrere Rechner, Racks
- o Schattenmaster
- o Rebalancing

■ Konsistenz

- o Locking
- o Checksum wird von jedem Chunkserver benutzt, um Datenfehler zu erkennen und zu beseitigen
- o Logging (zur Wiederherstellung des Masterzustands)

■ Effizienz

- o Chunk 64 MB
- o Parallel

■ Skalierbarkeit

WS 07/08

tien_duc.dinh@urz.uni-heidelberg.de

27

Checksum:

Wenn es eine Anforderung (z.B read) von Client gibt, prüft Chunkserver erstmal die zu liefernden Daten, ob sie fehlerfrei sind, bevor er sie zum Client schickt. Wenn die Datenbits bei einem Chunk nicht mit dem Checksum übereinstimmt, schickt Chunkserver eine Fehlermeldung zu Client und einen Bericht zu Master.

Client wird dann die Operation bei einem anderen Chunkserver versucht, auszuführen, während der Master den Chunk von einem anderen Replica klonet und dann eine Anweisung zu Chunkserver, der den Bericht geschickt hat, den neuen gültigen Chunk zu ersetzen. (-> man kann sich so vorstellen, Client bekommt von Master eine Liste von Chunkservern (abgekürzt CS) z.B. CS-1, CS-2, CS-3. Wenn Chunks auf CS-1 korrupt sind, versucht Client auf CS-2 die Operation auszuführen).

Checksum hat einen kleinen Effekt auf die read-Leistung, da die read-Daten meisten nur auf ein paar Blocks liegen, deshalb sind die Checksum-Bits nicht so groß. So ist die Leistung noch gewährleistet, während man die Zuverlässigkeit und Konsistenz gewinnen kann.

Was ist HDFS ?

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

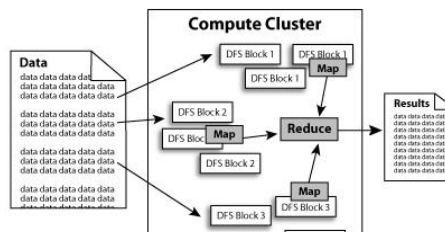
- o Architektur
- o Datenstruktur

o Operationen

o Systemeigenschaften

3. HDFS & Demo

- Hadoop ist eine Softwareplattform, mit der man leicht Applikationen schreiben und ausführen kann, die auf große Datenmengen prozessieren.
- Hadoop ist in Java geschrieben
- Hadoop implementiert Map-Reduce (wie bei Google) und hat HDFS (Hadoop Distributed File System) als Dateisystem
- HDFS hat Eigenschaften wie GFS, z.B. Zuverlässigkeit, Skalierbarkeit usw.
- Map-Reduce teilt die Daten in mehrere Blocks ein, berechnet einzeln und fügt dann alles wieder zum Endergebnis hinzu
- Hadoop momentan auf 2000 Knoten demonstriert - Ziel in Zukunft ist 10.000 !



WS 07/08

tien_duc.dinh@urz.uni-heidelberg.de

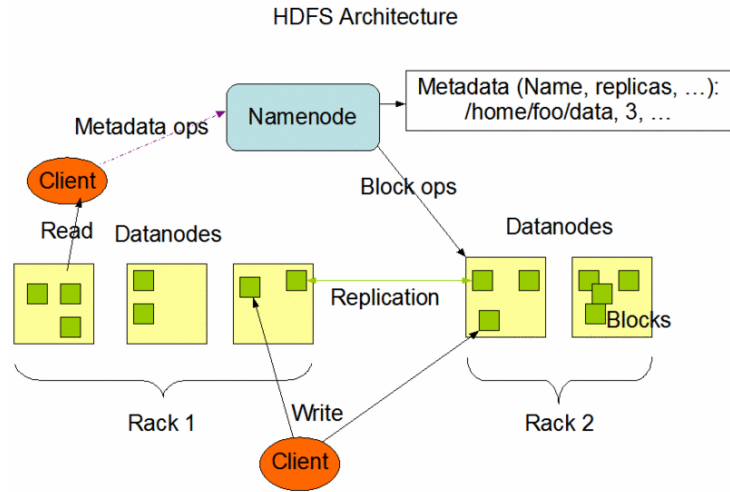
28

Hadoop ist sehr ähnlich wie Google, da Hadoop Konzepte von Google übernimmt.

Hadoop Architektur

Inhalt

- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- 3. HDFS & Demo



[Referenze]

Inhalt

1. Einleitung

- o Motivation
- o Grundbegriffe

2. Überblick

- o Architektur
- o Datenstruktur
- o Operationen
- o Systemeigenschaften

3. HDFS & Demo

- The Google File System (By Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung)
- http://en.wikipedia.org/wiki/Google_File_System
- <http://lucene.apache.org/hadoop/>
- <http://lucene.apache.org/hadoop/quickstart.html>
- http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_%28Single-Node_Cluster%29
- <http://wiki.apache.org/lucene-hadoop/HadoopMapReduce>

[Schluss]

Inhalt

- 1. Einleitung
 - o Motivation
 - o Grundbegriffe
- 2. Überblick
 - o Architektur
 - o Datenstruktur
 - o Operationen
 - o Systemeigenschaften
- ▶ 3. HDFS & Demo

Danke für Eure Aufmerksamkeit !