Verteilte Dateisysteme in der Cloud

Cloud Data Management

Maria Moritz

Inhalt

1.) Anforderungen an verteilte Dateisysteme

2.) GoogleFS

3.) Hadoop Distributed File System

4.) Amazon S3

5.) Zusammenfassung

Anforderungen an verteilte Dateisysteme

- große Menge an Daten organisieren und verarbeiten
- Datenverfügbarkeit
- Dauerhaftigkeit
- schnelle Bereitstellung der Daten
- Fehlertoleranz
- auf handelsübligen Maschinen lauffähig

GoogleFS

Aufbau

- Cluster besteht aus Master und mehreren Chunkservern

- Dateien in Chunks gesplittet und lokal auf Chunkservern gespeichert

- eindeutige Chunk-Handles vom Master zur Erstellungszeit bestimmt

- Replikationen der Chunks auf mehreren Chunkservern

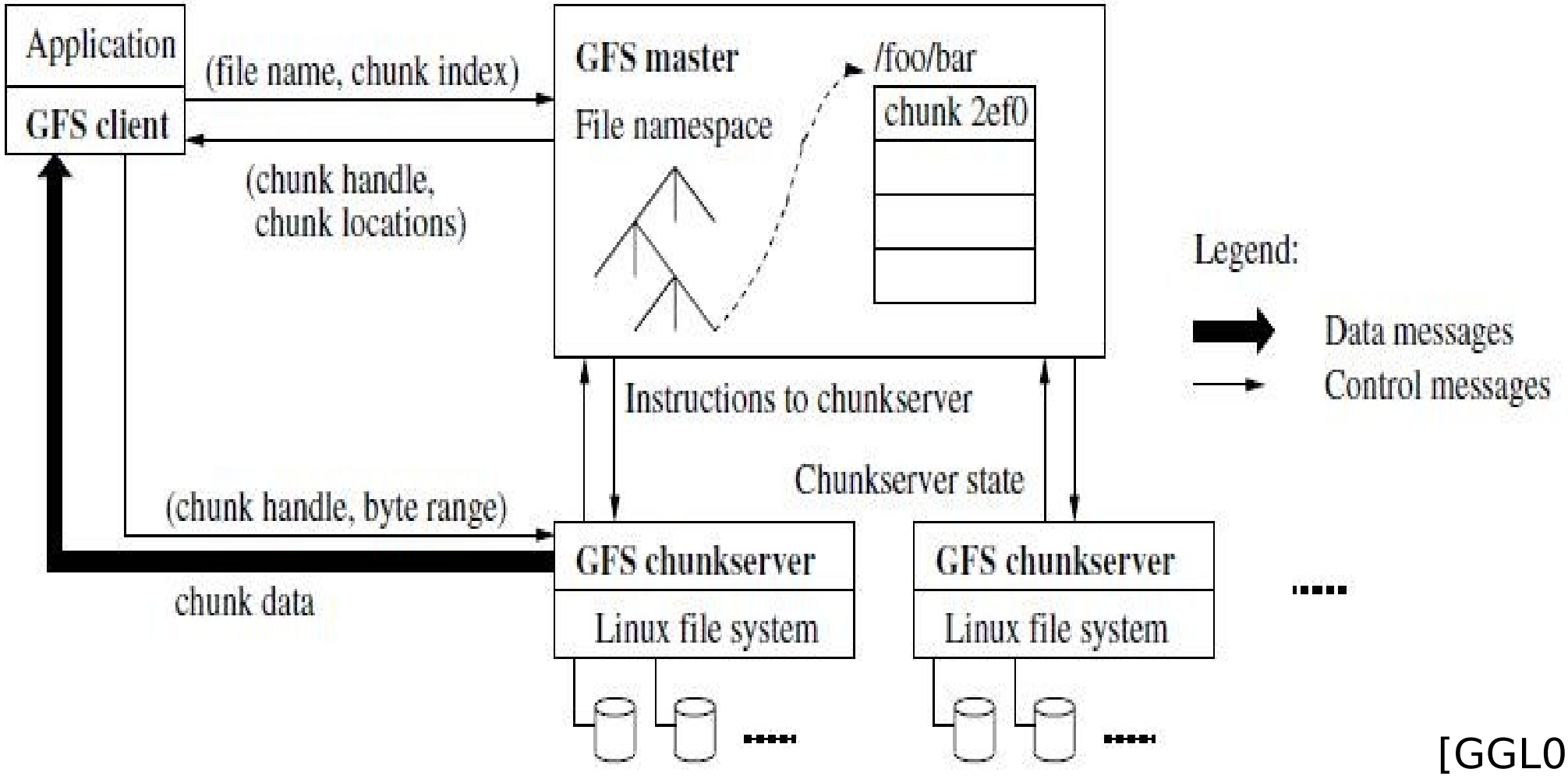
- Master organisiert alle Dateisystem-Metadaten

- verwaltet Namensraum uns Aufenthalt der Chunks und übernimmt Mapping von Dateien auf Chunks

- Kommunikation via Heartbeat-Messages

GoogleFS

Aufbau



[GGL03]

GoogleFS

Single Master

- Clients fragen Master nach Chunkservern, um Daten zu lesen/schreiben:

  > Client sendet Master Request mit Dateinamen und Chunkindex des Chunks
  > Master Antwortet mit Chunk-Handle und Ort der Replikationen
  > Client cached diese Dateinamen und Chunkindex

- nun Request an nächst liegenden Chunkserver

- solange Chunk-Informationen gecached, Anfragen an Chunkserver, ohne wieder mit Master kommunizieren zu müssen

- minmiert Master-Flaschenhals

GoogleFS

Konsistenzmodel

- Änderungen an Metadaten vom Master ausgeführt und in Operation-Log gespeichert

- Änderungen eines Chunks werden in selber Reihenfolge auf alle Replikationen angewendet

- Chunk-Versionsnummerierung um ungültige Replikationen zu finden

- Heartbeats um Datenverlust durch Komponentenausfälle zu vermeiden

GoogleFS

Schnappschüsse

- um Kopien von Verzeichnissen zu erstellen oder aktuellen Status des VZ zu sichern

- wenn Master Schnappschussanfrage erhält, entzieht er zunächst allen Chunks ausstehende Leases

- stellt sicher, dass folgende Schreiboperationen auf Chunks Interaktion mit Master erfordern und gibt dem Gelegenheit neue Kopie des Chunks zu erstellen

- dann dupliziert Master Metadaten für Quelldateinen und -verzeichnisse

- die neu erstellten Schnappschussdateien zeigen auf die gleichen Chunks wie Quelldateien

GoogleFS

Replikationsverwaltung

- bei Erstellung einer Replikation muss Master entscheiden, wo sie platziert werden soll, um Speicher optimal auszunutzen

- Chunk wird re-repliziert soblad Anzahl der Replikate unter bestimmten Wert fällt (z.B durch Chunk-Serverausfall oder benutzerspeziefische Erhöhung der Replikationszahl)

- aktive Dateien eher ersetzt, als bereits gelöschte Objekte

GoogleFS

Garbage-Collection

- auch Löschen einer Datei wird geloggt

- Datei zunächst mit versteckten Namen (und Löschzeitpunkt) versehen

- beim Scannen des Dateisystems werden ältere Dateien gelöscht,

- bis dahin kann Datei gelesen und wieder hergestellt werden

- beim endgültigen Löschen werden Links auf zugehörige Chunks gelöscht und in weitern Scan die "verwaisten" Chunks.

GoogleFS

Verfügbarkeit

Fast Recovery:

- sowohl Master als auch Chunk-Server können Status wieder herstellen, egal wie beendet wurden
- keine Unterscheidung zwischen normalem oder fehlerhaftem Abbruch
- Server routinemäßig durch Löschen des Prozesses ausgeschaltet

Chunk Replikation:

- Benutzer kann Replikations-Level auf den verschiedenen Bereichen des Namensraumes selbst bestimmen (Standart ist drei)
- Master repliziert Chunks, soblad Server offline gehen oder beschädigte Replikation melden

GoogleFS

Verfügbarkeit

Master Replikation:

- Operation-Log des Masters auf mehreren Maschinen repliziert

- Master-Prozess hat Verantwortung für Änderungen und Hintergrund-Aktivitäten (Garbage-Collection)

- wenn er ausfällt, kann er sofort neu starten

- wenn dessen Platte oder Rechner ausfällt startet Monitoring-Infrastructur auf anderer Maschine neuen Master-Prozess

Hadoop Distributed File System (HDFS)

- fehlertolerantes, verteiltes Dateisystem

- wurde entwickelt um auf handelsüblicher Hardware zu laufen

- schnellen Zugang zu Anwendungen mit großen Datensätzen

- wurde ursprünglich für Apache-Nutch, einer Open-Source-Suchmaschine entwickelt

Hadoop Distributed File System (HDFS)

Architektur

- Master-Slave-Architektur

- Cluster besteht NameNode und DataNodes

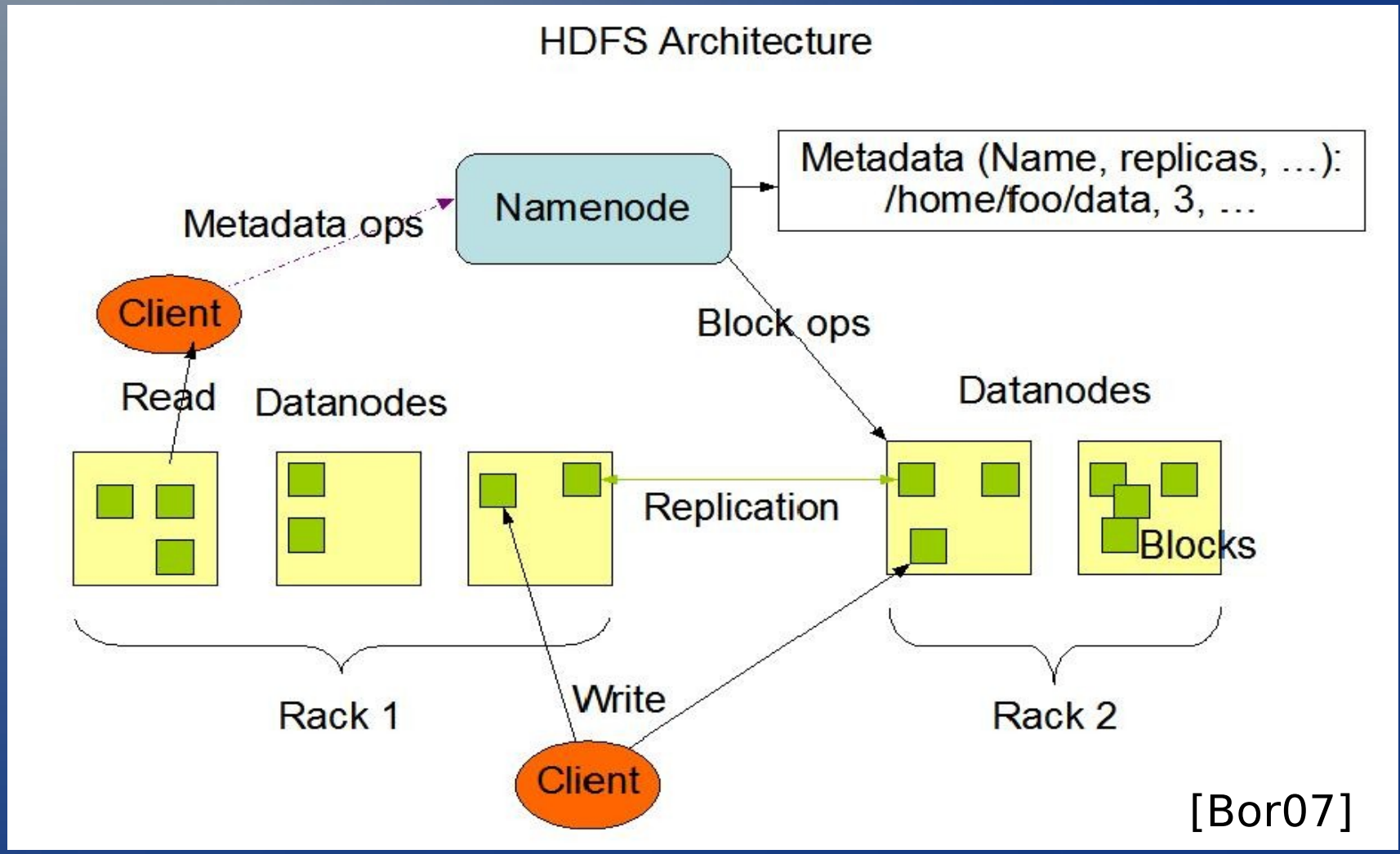
- NameNode:

  > organisiert Namensraum des Dateisystems
  > führt Operationen wie Öffnen, Schließen, Umbenennen von Dateien aus
  > regelt Zugang zu den Dateien durch die Clients
  > mappt Datenblöcke auf DataNodes

- DataNodes:

  > speichern Daten der ihnen anhaftenden Knoten
  > bedienen Lese- und Schreiboperationen der Clients
  > führen Blockerstellung und -löschen und
  > Replikationen im Auftrag der NameNodes durch

Hadoop Distributed File System (HDFS)



Hadoop Distributed File System (HDFS)

Datenreplikation

- Dateien werden als Block-Sequenz abgespeichert, diese werden repliziert

- Dateien werden einmalig geschrieben und besitzen auch nur einen Schreiber zur Zeit

- NameNode bekommt periodische „Heartbeats“ und Blockrepots, die Liste aller Blöcke des DataNodes beinhalten, somit Kenntnis über das Funktionieren des Knotens

Hadoop Distributed File System (HDFS)

Replikationsersetzung

- größere HDFS-Instanzen laufen auf Computer-Cluster, das über mehrere Racks verteilt ist

- NameNode bestimmt Rack-ID, die jeder DataNode enthält

- Standard des Replikationsfaktors ist drei

- davon zwei auf unterschiedlichen Knoten des selben Racks

- eine dritte auf einem DataNode eines separaten Racks

- somit wird Schreibverkehr zwischen den Racks reduziert, und trotzdem die Datenverfügbarkeit garantiert, da Rack-Ausfall seltener als Knotenausfall

Hadoop Distributed File System (HDFS)

Persistenz der Metadaten

- EditLog als Transaktionslog, um Änderungen auf Metadaten zu speichern

- FsImage-Datei enthält Namensraum und Mapping der Dateiblöcke auf Dateien

- beides beim NameNode gespeichert

- beim Starten des NameNodes wird

  > FsImage und EditLog von der Platte gelesen
  > Transaktionen des EditLogs zum FsImage hinzu geführt
  > diese neue Version des FsImages auf Platte gespeichert

- beim Starten eines DataNodes wird Liste aller Datenblöcke des lokalen Dateisystem erstellt und als Blockreport an NameNode gesendet

Hadoop Distributed File System (HDFS)

Robustheit

- häufige Fehler sind:
  - > NameNode-Asufälle
  - > DataNode-Ausfälle
  - >Netzwerk-Partitionen

- DataNode-Ausfälle:
  - > durch regelmäßigen Heartbeat wird Knotenausfall festgestellt
  - > diese werden als tot markiert, bekommen keine I/O-Requests mehr
  - > NameNode findet neu zu replizierenden Blöcke und initiiert Replikationen

- NameNode-Ausfälle:
  - > FsImage und EditLog sind zentrale Datenstrukturen von HDFS
  - > deshalb muss Unversehrtheit gesichert werden
  - > zu diesem Zweck kann NameNode mehrere Kopien dieser verwalten

Amazon S3

Architekturüberblick

- S3 speichert Daten in Buckets (ähnlich Ordnern – Gebührenerhebung)

- Buckets speichern unbegrenzt viele Datenobjekte

- Objekte besteht aus *Namen, Daten und Metadaten*

- Benutzer können Objekte lesen, schreiben oder verändern

- Umbenenn oder Verschieben eines Objektes bedeutet Herunterladen und neu Hochladen desselben unter neuen Namen

- Gebühren für:
    > *Speichern*
    > *Transferieren (Upload und Download)*
    > *Operationen (get, put, list)*

Amazon S3

Architekturüberblick

- Benutzer registrieren sich bei Amazon Web Services mit Public und Private-Key

- Jedes S3-Account für Zahlungszwecke mit Kreditkarten-Nummer verbunden

- Buckets können konfiguriert werden, um Zugangs-Log-Einträge mit Objekt, Zeit und Request-Typ zu speichern

- Datenzugang via SOAP, BitTorrent

Amazon S3

Evaluation - Datenverfügbarkeit

S3-Bucket mit Testobjekten verschiedener Größe (1KB, 1MB, 16MB, 100MB) :

> Tests auf EC2-Cluster, um Bandbreite von S3 zu ermitteln
> während Testlauf Daten zufällig geschrieben und gelesen

von 107.556 Tests ergaben fünf Instanzen eine Widerholung des HTTP PUT-Requests, 23 ergaben leeren Antwort-Code

nur fünf Lesezugriffe erforderten Wiederholung

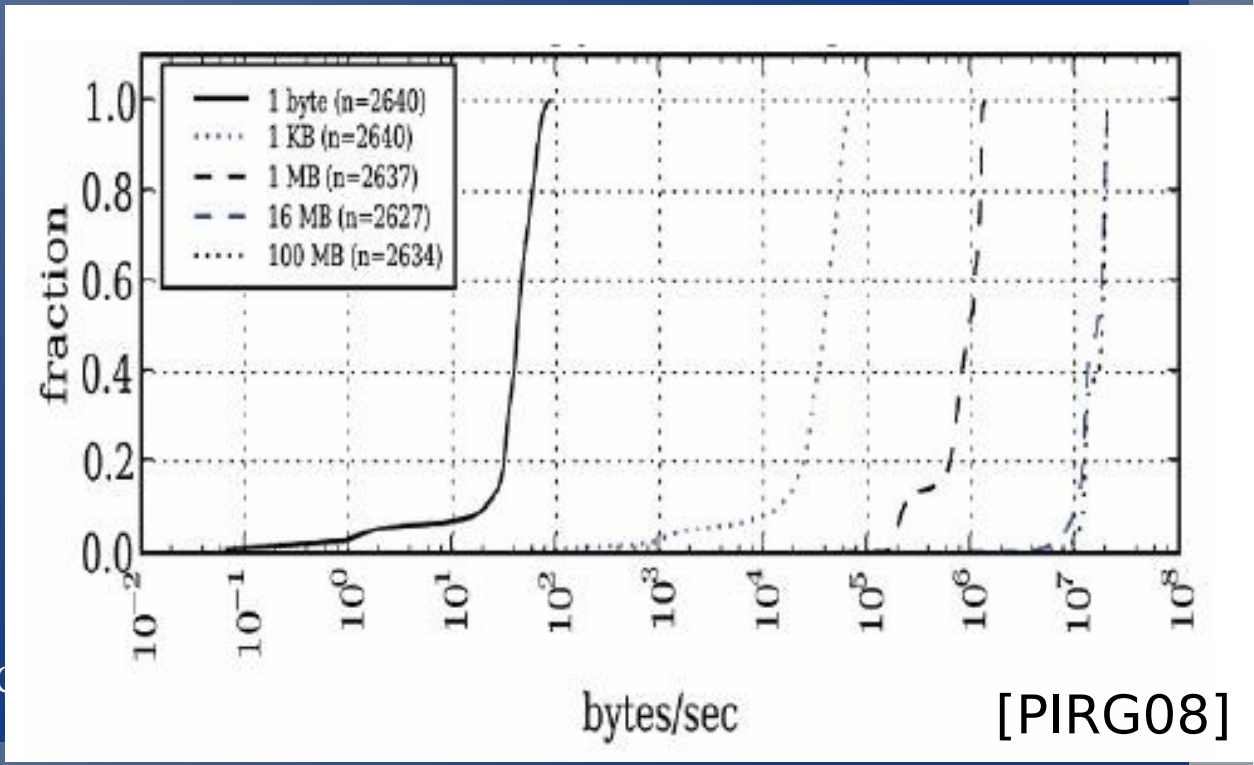
Amazon S3

Datenzugriffsperformanz

Download-Geschwindigkeit von Dateien (1B, 1KB, 1MB, 16MB, 100MB):

> kleine Objekte haben großen Transaktions-Overhead
> maximal 100 Transactionen pro Sekunde
> max. Bandbreite ca. 21 MB/s

Uploads verhalten sich ähnlich



[PIRG08]

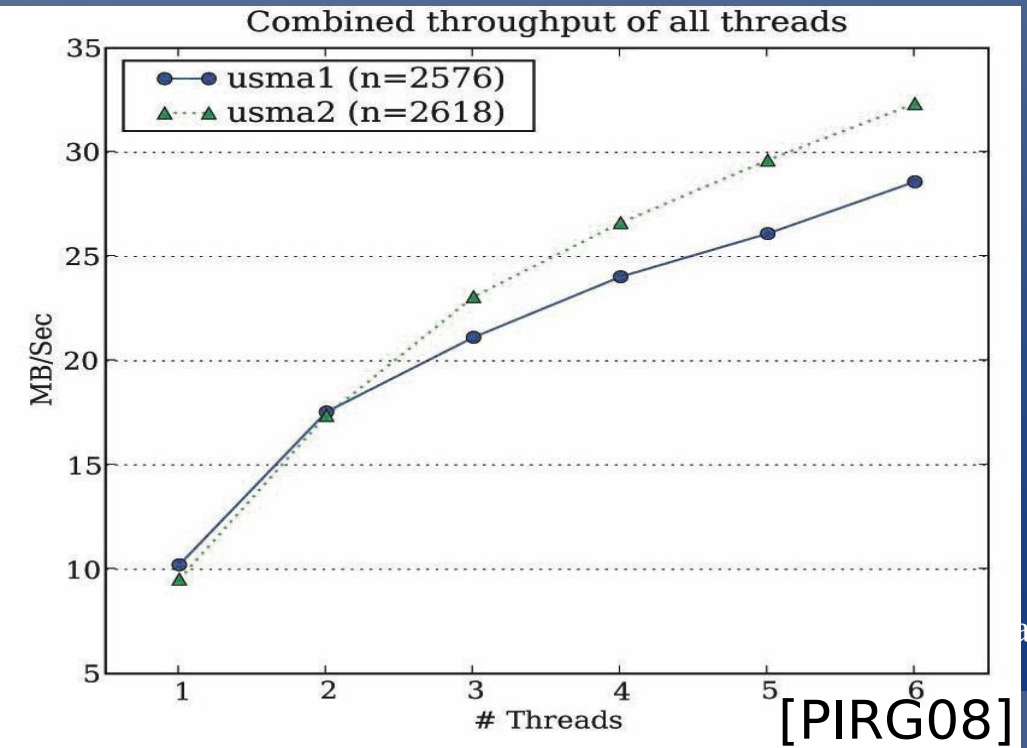
Amazon S3

Konkurrente Performance

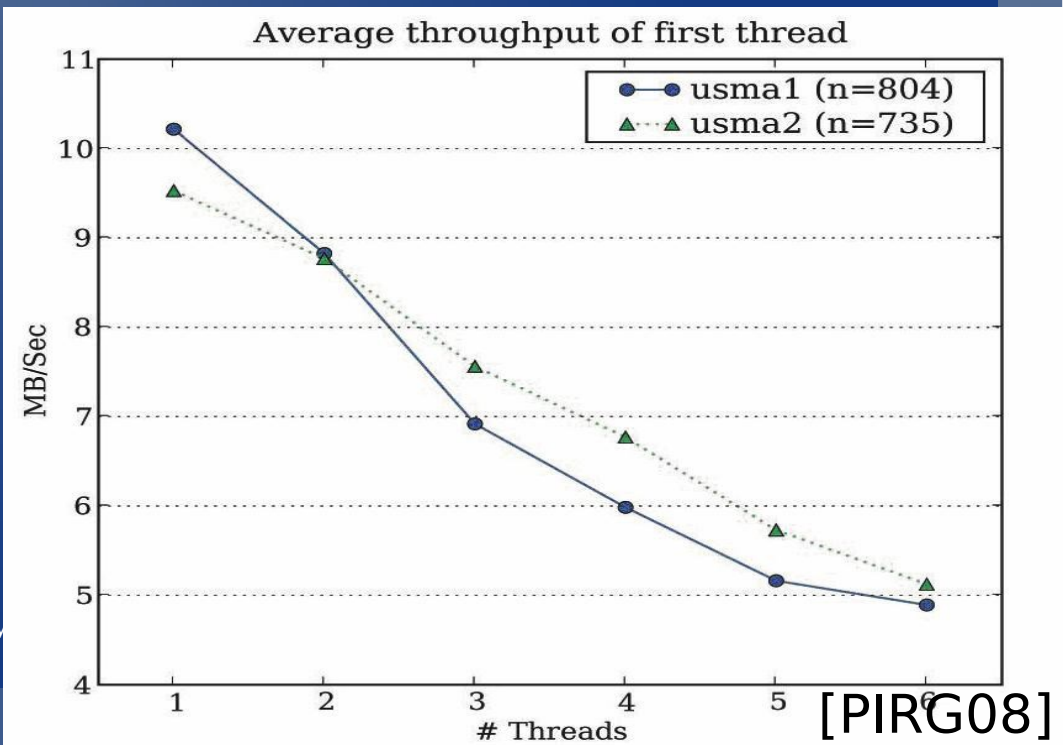
- Versuch mit wiederholtem Zugriff zweier virtueller Maschinen verschiedener EC2-Cluster auf Daten desselben Buckets

- während 11 Stunden wurde pro VM 10 min, jeweils ein Thread, dann zwei usw. mit GET- und PUT-Operationen von 100MB gestartet

- mit Zunahme Threads nimmt Bandbreite pro Thread ab, aber Gesamt-Bandbreite zu



[PIRG08]

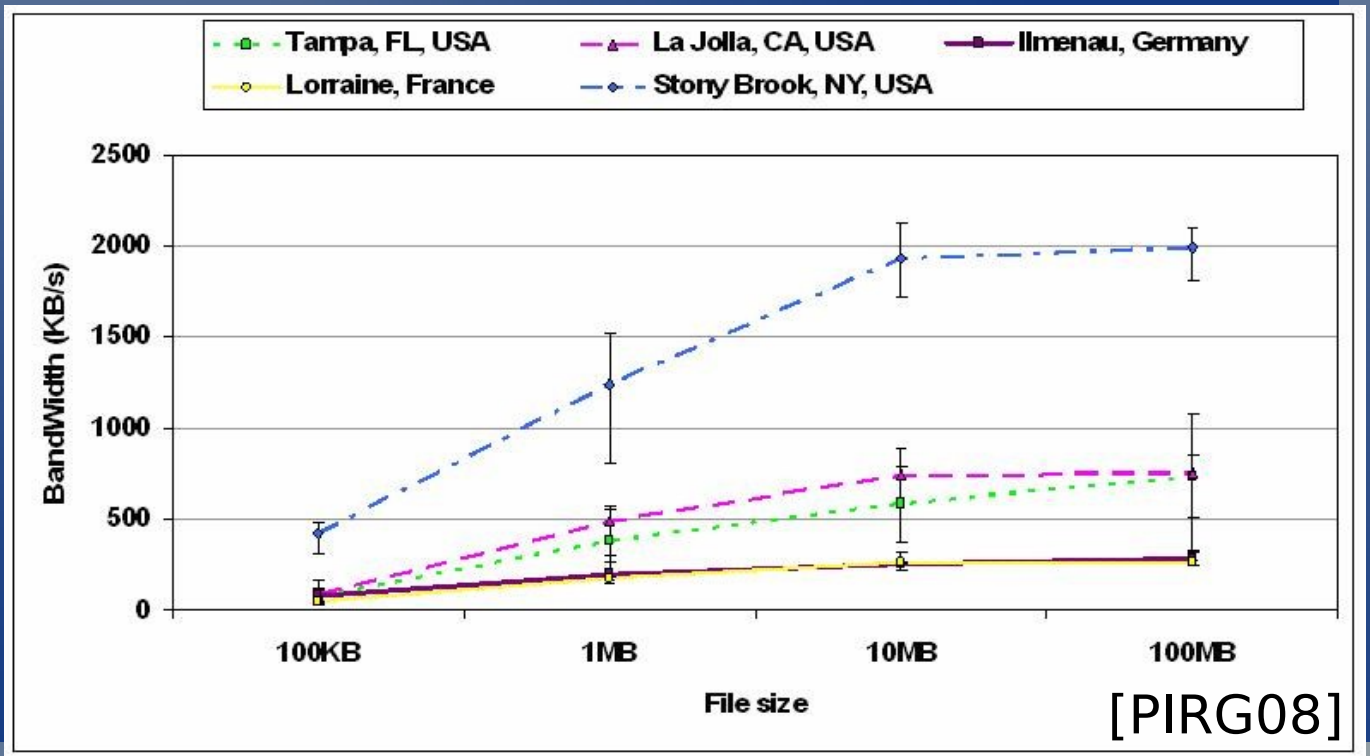


[PIRG08]

Amazon S3

Remote Access Performance

- Zugriffszeiten verschiedener Standorte
- 28 Experimente über 7 Tage
- Durchsatz von Standort abhängig



[PIRG08]

Zusammenfassung

	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 Datenfluus 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

Quellen

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

- [GGL03]: Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung, The Google File System, ACM New York, 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? ACM New York, 2008,
http://portal.acm.org/citation.cfm?id=1383526.