

## Verwaltung großer Datenmengen

Motivation  
Google File System  
Bigtable  
Colossus

## ■ CAP-Theorem nach [Brewer]

- In einem verteilten System ist es nicht möglich gleichzeitig

- Konsistenz (*Consistency*)
- Verfügbarkeit (*Availability*)
- Partitionstoleranz (*Partition Tolerance*)

zu garantieren

- **Abschwächung (mindestens) einer der Eigenschaften erforderlich**

## ■ Herausforderungen

- Welche der Eigenschaften sollen in welchem Umfang garantiert werden?
- Wie lassen sich **Speichersysteme speziell auf Anwendungen zuschneiden**?

## ■ Literatur



Eric A. Brewer

**Towards robust distributed systems**

*Proceedings of the 19th Symposium on Principles of Distributed Computing (PODC '00), S. 7, 2000.*

# Google File System

## ■ Einsatzszenario

- Verwendung hunderter bzw. tausender Rechner
- Verwaltung sehr großer Datenmengen
- **Hardware-/Software-Ausfälle sind keine Ausnahme**, sondern die Regel

## ■ Google File System

- Auf eine spezielle Kategorie von Anwendungen zugeschnitten
- **Fehlertoleranz durch Replikation**
- Ausgelegt auf Einsatz von Commodity-Hardware
- **Abgeschwächtes Konsistenzmodell**
- Vorbild für das quelloffene *Hadoop Distributed File System (HDFS)*

## ■ Literatur



Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

**The Google file system**

*Proceedings of the 19th Symposium on Operating Systems Principles (SOSP '03), S. 29–43, 2003.*

# Anforderungen der Anwendungen an das Dateisystem

## ■ Dateigröße

- Fokus auf **sehr große Dateien** [→ Mehrere Gigabytes pro Datei.]
- Kleine Dateien sollen unterstützt werden, sind jedoch nicht vorrangig

## ■ Zugriffsmuster

- Lesezugriffe
  - **Lesen großer, oftmals zusammenhängender Bereiche** einer Datei
  - Lesen kleiner Teilbereiche einer Datei, dazwischen Sprünge
- Schreibzugriffe
  - Schreibanfragen hängen in der Regel Daten an eine Datei an
  - **Wahlfreies Schreiben ist die Ausnahme**, muss jedoch unterstützt werden

## ■ Weitere Eigenschaften

- Eine einmal geschriebene Datei wird meistens nicht mehr modifiziert
- **Paralleles Anhängen an dieselbe Datei** durch mehrere Prozesse ist häufig
- Hoher Durchsatz wichtiger als kurze Antwortzeiten für einzelne Anfragen

## Schnittstelle

### Hierarchische Datenverwaltung

- Verzeichnisse
- Dateien

### An traditionelle Dateisysteme angelehnte Schnittstelle

| Operation | Beschreibung                      |
|-----------|-----------------------------------|
| create    | Anlegen einer Datei               |
| delete    | Löschen einer Datei               |
| open      | Öffnen einer Datei                |
| close     | Schließen einer Datei             |
| read      | Lesen von Daten aus einer Datei   |
| write     | Schreiben von Daten in eine Datei |

### Zusätzliche Operationen

|          |   |
|----------|---|
| append   | Atomares Anhängen von Daten an eine Datei |
| snapshot | Erstellen eines Sicherungspunkts          |

## Architektur

### Grundlegender Ansatz

- Aufteilung großer Dateien in **Datenblöcke fester Größe** (*Chunks*)
  - Typische Größe: 64 MB
  - Eindeutig identifizierbar durch *Chunk-Handles* (jeweils 8 Bytes)
- Redundantes Speichern eines Datenblocks auf mehreren Rechnern

### Zentrale Komponenten

- Chunk-Server
  - Verwaltung von Datenblöcken
  - **Persistente Datenspeicherung** auf der lokalen Festplatte
- Master-Server
  - Verwaltung von Metadaten im Hauptspeicher
    - \* **Zuordnung von Datenblöcke auf Dateien**
    - \* Speicherorte von Datenblöcken (= Chunk-Server)
    - \* Zugriffsberechtigungen von Clients
  - Zusätzlich: Sicherung zentraler Metadaten in einer replizierten Log-Datei
  - **Überwachung von Chunk-Servern mittels „HeartBeat“-Nachrichten**
  - Koordinierung der Lastverteilung

## Schreiboperationen

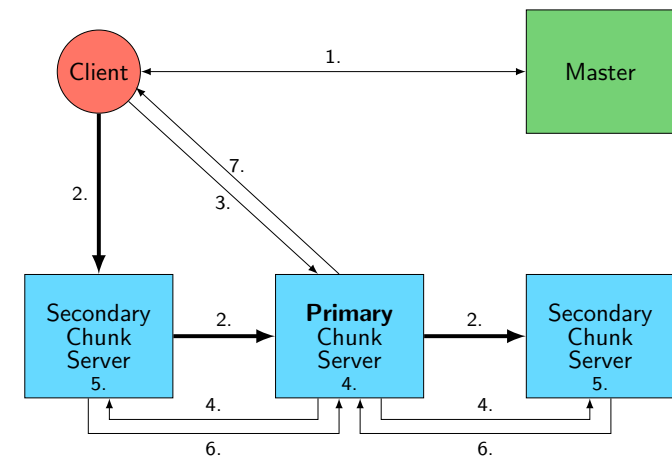
### Vorbereitungen beim Anlegen eines Datenblocks

- Master wählt drei für den Datenblock zuständige Chunk-Server aus
- Per Lease: Ernennung eines der Server zum *Primary*
- Die anderen beiden Server übernehmen die Rolle von *Secondaries*

### Vorgehensweise bei Schreib Anfragen auf Datenblöcken

1. Ermittlung der für den Datenblock zuständigen Chunk-Server
  - Anfrage des Clients an den Master
  - Speicherung der Master-Antwort in lokalem Client-Cache
2. **Verteilung der Nutzdaten** und Warten auf Empfangsbestätigungen
  - Weitergabe zum jeweils „nächstgelegenen“ Chunk-Server
  - Auf IP-Adressen basierende Metrik zur Auswahl des nächsten Empfängers
- **Durchführung der Schreiboperation**
  3. Senden des eigentlichen Schreibkommandos vom Client an den Primary
  4. Ausführung auf dem Primary und Weiterleitung an Secondaries
  5. Ausführung der Schreiboperation auf den Secondaries
  6. Primary: Sammlung der Bestätigungen aller zuständigen Chunk-Server
  7. Senden einer Erfolgsmeldung vom Primary an den Client

## Schreiboperationen



## Weitere Operationen

- Schreibaufruf mittels append-Operation
  - **Atomares Anhängen von Daten** an eine Datei
  - Typischer Anwendungsfall
    - Gleichzeitiger Zugriff durch mehrere Clients
    - Paralleles Schreiben in dieselbe Datei
  - Unterschiede zur normalen Schreiboperation
    - **Primary legt Offset im Datenblock fest**
    - Falls die Daten nicht mehr in den aktuellen Datenblock passen
      - \* Primary weist Secondaries an, den Datenblock mit Padding-Bytes zu füllen
      - \* Client muss Anhängoperation auf dem nächsten Datenblock wiederholen
    - Primary teilt dem Client den tatsächlichen Speicher-Offset mit
- Ablauf einer Leseoperation
  1. Client fragt Master nach für den Datenblock zuständigen Chunk-Servern
  2. Client speichert Master-Antwort für spätere Anfragen in lokalem Cache
  3. Client sendet **Leseanfrage zum „nächstgelegenen“ Chunk-Server**



## Umgang mit Fehlersituationen

- Mechanismen zur effizienten Behandlung von Master-Ausfällen
  - Replikation des Zustands über mehrere Rechner
  - Relativ kleiner Master-Zustand → Schneller Neustart möglich
  - **Einsatz von Shadow-Master-Servern** für nichtmodifizierende Anfragen
- Behandlung von Chunk-Server-(Teil-)Ausfällen
  - Datenkorruption
    - **Fehlerdetektion bei Leseanfragen**
      - \* Integritätsüberprüfung der gespeicherten Daten durch Chunk-Server
      - \* Nutzung von Checksummen über 64 KB-Blöcke
    - Meldung an den Master bei der Erkennung von Fehlern
    - Master leitet Erstellung eines neuen Replikats ein
  - Rechnerausfall
    - HeartBeat-Nachrichten an den Master bleiben aus
    - Master leitet Erstellung neuer Replikate ein
    - Master bestimmt **nach Ablauf der Leases neue Primaries** für Datenblöcke



## Umgang mit Fehlersituationen

- Fehler bei der Bearbeitung von Anhängoperationen  
[Vergleichbare Probleme können auch beim wahlfreien Schreiben auftreten.]
  - Mindestens ein Chunk-Server sendet keine Erfolgsbestätigung
  - Client erhält eine Fehlermeldung

→ **Client wiederholt die komplette Anhängoperation**
- Abgeschwächte Konsistenzeigenschaften
  - Erfolgreiche Bestätigung einer Anhängoperation: Garantie, dass der Datensatz auf den Chunk-Servern **am selben Offset** gespeichert wurde
  - Potentielle Auswirkungen von Fehlern
    - Inhalt eines Datenblocks kann zwischen den Replikaten divergieren
    - Ein von der Anwendung einmalig angehängter Datensatz kann mehrfach vorliegen
- Auswirkungen auf Anwendungen
  - Anwendungen müssen mit den schwächeren Garantien umgehen können
  - Beispiele für mögliche Maßnahmen
    - Einsatz von selbstverifizierenden, selbstidentifizierenden Datensätzen
    - Berechnung von Checksummen durch die Anwendung



## Zentraler Master – Designentscheidung im Rückblick


- Vorteil: **Vereinfachung des Designs** ermöglichte schnelle Umsetzung
- Probleme
  - Unvorhergesehene Einsatzszenarien (z. B. Gmail)
    - Nicht alle auf dem System betriebenen Anwendungen sind stapelorientiert
    - Chunk-Größe von **64 MB ist für manche Anwendungen zu groß**
  - Symptome
    - Metadatenverwaltung für viele kleine Dateien → **Master als Flaschenhals**
    - Master-Ausfall problematisch für latenzsensitive Anwendungen
- Konsequenzen
  - **Verwendung mehrerer Instanzen** pro Datenzentrum
  - Entwicklung von Systemen mit verteiltem Master (z. B. *Colossus*)
- Literatur
  - Marshall Kirk McKusick and Sean Quinlan  
**Evolution on fast-forward**  
*Queue – File Systems*, 7(7):10–20, 2009.



## Bigtable

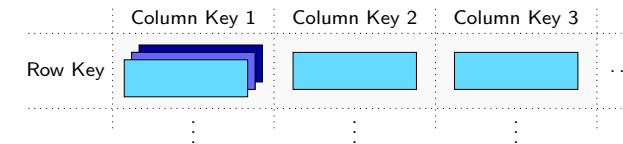
- Unterstützung auf **strukturierten Daten** basierender Anwendungen
  - Erstellung und Verwaltung eines Web-Index
  - Datenspeicher für Google Maps/Earth
  - ...
- Implementierung
  - Architektur
    - Zugriff per Client-Bibliothek
    - Steuerung des Systems durch einen **zentralen Master-Server**
    - Skalierbarkeit durch Bearbeitung von Datenzugriffen auf zusätzlichen Servern
  - **Persistente Speicherung der Daten im Google File System**
  - Vorbild für das quelloffene *HBase*

### Literatur

 Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson Hsieh, Deborah A. Wallach, Mike Burrows et al.  
**Bigtable: A distributed storage system for structured data**  
*Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06)*,  
 S. 205–218, 2006.

## Datenmodell

- Verwaltung von **Daten in Tabellenform**
  - Speicherung mehrerer Versionen pro Zelle
    - Identifizierung mittels System- oder Anwendungszeitstempeln
    - Kriterien für automatisierte Garbage-Collection
      - \* Maximum für die Anzahl der pro Zelle gespeicherten Versionen
      - \* Obergrenze für das Alter einer Version
  - **Atomare Schreib- und Lesezugriffe** auf Zeilen

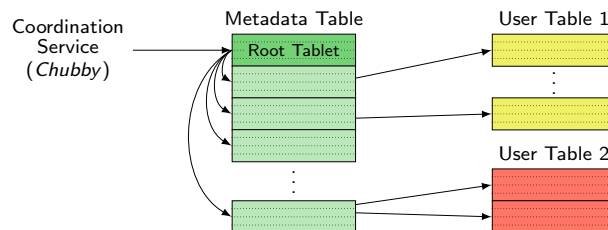


- Skalierbarkeit durch **Partitionierung von Tabellen**
    - **Tablet**: Zusammenhängender Abschnitt benachbarter Zeilen
    - Zuordnung verschiedener Tablets zu unterschiedlichen Servern
- Oft gemeinsam gelesene Einträge sollten benachbarte Schlüssel haben

## Implementierung

- Hierarchie
  - Metadaten-tabelle: Informationen zu Tablets von Nutzertabellen
  - **Root-Tablet**: Verwaltung der Struktur der Metadaten-tabelle
  - Koordinierungsdienst: Speicherung eines Zeigers auf das Root-Tablet

[→ Details in späterer Vorlesung.]

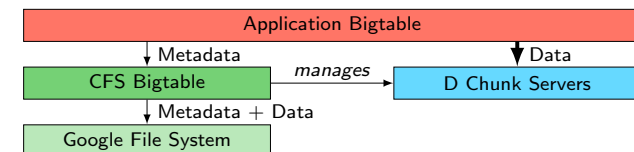


- Einsatz des Google File Systems zur persistenten Speicherung
  - **Sicherungspunkte** von Tablet-Zuständen
  - Log-Dateien mit den **Modifikationen seit dem letzten Sicherungspunkt**

[Hinweis: Der Ansatz ist sehr ähnlich zur Implementierung von Tabellen im Partition Layer von Microsoft Azure Storage.]

## Colossus

- Colossus File System (CFS): **Google-internal Nachfolger des Google File Systems**
  - Details über Aufbau und Funktionsweise nicht veröffentlicht
  - Chunk-Größe konfigurierbar (Standardwert: 1 MB)
  - Skalierbarkeit durch Hierarchie von Colossus-Instanzen
  - **Getrennte Speicherung von Meta- und Nutzdaten**
    - *CFS Bigtable*: Verwaltung des Dateisystems
    - *D*: Speicherdienst für Datenblöcke



### Literatur

 Denis Serenyi  
**Cluster-level storage @ Google**  
*Keynote at the 2nd Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Intensive Computing Systems, 2017.*