# Überblick





# Datenverwaltung in der Cloud

- CAP-Theorem nach [Brewer]
  - In einem verteilten System ist es nicht möglich gleichzeitig
    - Konsistenz *(**C**onsistency)*
    - Verfügbarkeit *(**A**vailability)*
    - Partitionstoleranz *(**P**artition 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**
  *Proc. 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
  - Einsatz von Commodity-Hardware
  - Fehlertoleranz durch Replikation
  - 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 und einer replizierten Log-Datei
      - Zuordnung von Datenblöcken auf Dateien
      - Speicherorte von Datenblöcken ($\rightarrow$ Chunk-Server)
      - Zugriffsberechtigungen von Clients
    - Überwachung von Chunk-Servern mittels *HeartBeat*-Nachrichten
    - Koordinierung der Lastverteilung

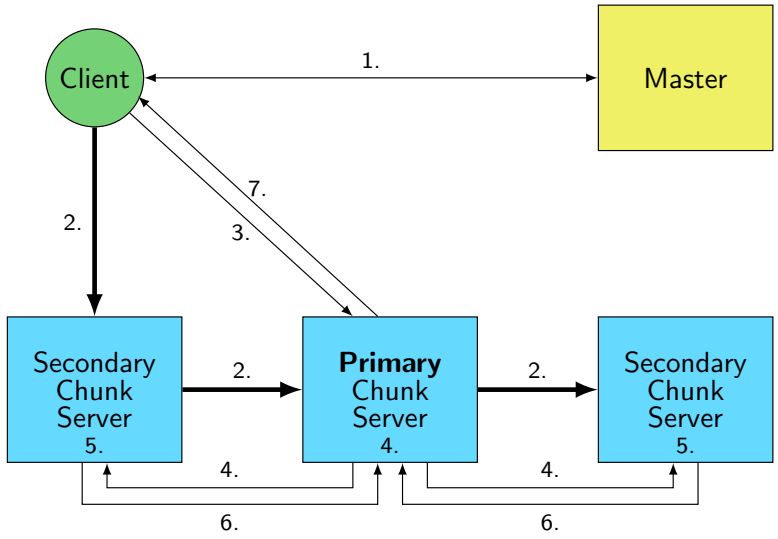


## Schreiboperationen

- Vorbereitungen beim Anlegen eines Datenblocks
  - Master wählt drei für den Datenblock zuständige Chunk-Server aus
  - Einer der Server wird vom Master per Lease zum *Primary* ernannt, alle anderen Server übernehmen die Rolle von *Secondaries*
- Vorgehensweise bei Schreibanfragen auf Datenblöcken
1. Client fragt Master nach für den Datenblock zuständigen Chunk-Servern, Client speichert Master-Antwort für spätere Anfragen in lokalem Cache
2. Client sendet Nutzdaten zum „nächstgelegenen“ Chunk-Server, von wo aus sie an die anderen Chunk-Server verteilt werden
   [Hinweis: Die Auswahl des „nächstgelegenen“ Server erfolgt mit Hilfe einer auf IP-Adressen basierenden Metrik.]
3. Sobald alle Chunk-Server den Empfang der Daten bestätigt haben, sendet der Client das eigentliche Schreibkommando an den Primary
4. Primary führt Operation aus und leitet Kommando an Secondaries weiter
5. Secondaries führen Schreiboperation aus
6. Primary sammelt Bestätigungen aller zuständigen Chunk-Server
7. Primary sendet Erfolgsmeldung an Client



## Schreiboperationen




## Weitere Operationen

- append-Operation
  - Atomares Anhängen von Daten an eine Datei
  - Typischer Anwendungsfall: 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 aktuellen Datenblock zu verschließen
      - Client muss Anhängeoperation 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 (→ Log-Datei)
  - Relativ kleiner Master-Zustand → Schneller Neustart möglich
  - Einsatz von *Shadow-Master-Servern* für nichtmodifizierende Anfragen
- Behandlung von Chunk-Server-(Teil-)Ausfällen
  - Datenkorruption
    - Bei Leseanfragen: Chunk-Server überprüfen die Integrität der gespeicherten Daten mittels Checksummen über 64 KB-Blöcke
    - Falls Fehler erkannt werden → Meldung an den Master
    - Master leitet Erstellung eines neuen Replikats ein
  - Rechnerausfall
    - HeartBeat-Nachrichten an den Master bleiben aus
    - Master leitet Erstellung eines neuen Replikats ein
    - Master bestimmt nach Ablauf der Leases neue Primaries für Datenblöcke
    - Erkennung veralteter Datenblockversionen durch Einsatz von Lease-Epochen



## Umgang mit Fehlersituationen

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

  → Client wiederholt die komplette Anhängeoperation
- Abgeschwächte Konsistenzeigenschaften
  - Erfolgreiche Bestätigung einer Anhängeoperation: Garantie, dass der Datensatz auf den Chunk-Servern am selben Offset gespeichert wurde
  - Potentielle Auswirkungen von Fehlern
    - Der 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

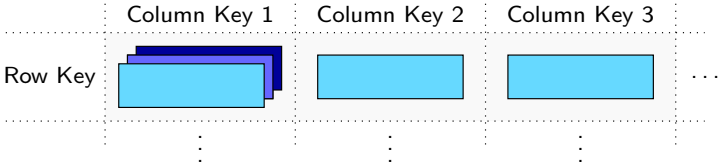
- Unterstützung auf strukturierten Daten basierender Anwendungen
  - Erstellung und Verwaltung eines Web-Index
  - Datenspeicher für Google Maps/Earth
  - ...
- Implementierung
  - Architektur
    - Zugriff per Client-Bibliothek
    - Steuerung des Systems durch einen zentralen Master-Server
    - Skalierbarkeit 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 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



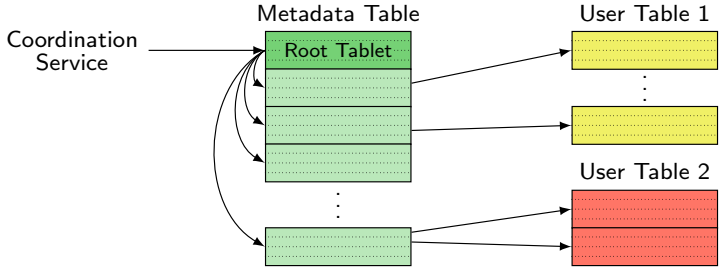

- Skalierbarkeit durch Partitionierung von Tabellen
  - *Tablet*: Zusammenhängender Abschnitt benachbarter Zeilen
  - Zuordnung verschiedener Tablets zu unterschiedlichen Servern

→ Oft gemeinsam gelesene Einträge sollten benachbarte Schlüssel haben



# Implementierung

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




- 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 Windows Azure Storage.]



# Zusammenfassung

- Google File System
  - Ausrichtung auf Anwendungen mit bestimmten Zugriffsmustern
    - Lesen großer, zusammenhängender Bereiche einer Datei
    - Schreiben von Datensätzen durch Anhängen
  - Fehlertoleranz durch Replikation
  - Abgeschwächtes Konsistenzmodell
    - Im Fehlerfall keine Garantie für starke Konsistenz
    - Spezielle Vorkehrungen auf Anwendungsebene erforderlich
  - Zentraler Master-Server kann zum Flaschenhals werden
- Bigtable
  - Verwaltung strukturierter Daten in Tabellenform
  - Skalierbarkeit durch Partitionierung von Tabellen
  - Persistente Speicherung der Daten im Google File System
- Verwandte quelloffene Systeme: HDFS und HBase (*Apache Hadoop*)