Middleware – Cloud Computing

Verwaltung großer Datenmengen

Wintersemester 2020/21

Tobias Distler

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)

Lehrstuhl für Verteilte Systeme und Betriebssysteme

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Überblick

Verwaltung großer Datenmengen
- Motivation
- Google File System
- Bigtable
- Colossus

Datenverwaltung in der Cloud

- **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
Proc. of the 19th Symp. on Principles of Distributed Computing (PODC '00), S. 7, 2000.

Überblick

Verwaltung großer Datenmengen
- Motivation
- Google File System
- Bigtable
- Colossus

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
  Proc. 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 Gigabyte pro Datei.)
  - Kleine Dateien sollen unterstützt werden, sind jedoch nicht vorrangig
- Zugriffsmuster
  - Schreibzugriffe
    - Schreibanfragen hängen in der Regel Daten an eine Datei an
    - **Paralleles Anhängen an dieselbe Datei** durch mehrere Prozesse ist häufig
    - Wahlfreies Schreiben ist die Ausnahme, muss jedoch unterstützt werden
  - Lesezugriffe
    - **Lesen großer, oftmals zusammenhängender Bereiche** einer Datei
    - Lesen kleiner Teilbereiche einer Datei, dazwischen Sprünge
- Weitere Eigenschaften
  - Eine einmal geschriebene Datei wird meistens nicht mehr modifiziert
  - 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

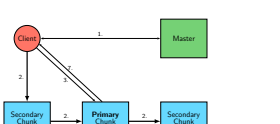
Architektur

- 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)
    - 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 **Schreibanfragen 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 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
  - Relativ kleiner Master-Zustand → Schneller Neustart möglich
  - **Einsatz von Shadow-Master-Servern** für nichtmodifizierende Anfragen
- Behandlung von Chunk-Server-(Teil-)Ausfallen
  - 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 Anhängeoperationen
  [Möglichkeiten: Problemen können auch beim Schreiben Schreiben.]
  - Die Erfolgsbestätigung von mindestens einem Chunk-Server kommt nicht an
  - Client erhält eine Fehlermeldung
  - → **Client wiederholt die komplette Anhängeoperation**
- Abgeschwächte Konsistenzgeschaften
  - Erfolgreiche Bestätigung einer Anhängeoperation: Garantie, dass der Datensatz auf den Chunk-Servern **am selben Offset** gespeichert wurde
  - Potentielle Auswirkungen von Fehlern
    - Ein von der Anwendung einmalig angehängter Datensatz kann mehrfach vorliegen
    - Inhalt eines Datenblocks kann zwischen den Replikaten divergieren
- **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

Überblick

Verwaltung großer Datenmengen
Motivation
Google File System
Bigtable
Colossus

Bigtable

- Unterstützung auf **strukturierten Daten** basierender Anwendungen (Beispiele)
  - 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 et al. **Bigtable: A distributed storage system for structured data** *Proc. 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 [→ Separate 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.]

Überblick

Verwaltung großer Datenmengen
Motivation
Google File System
Bigtable
Colossus

Google File System: 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
    - Master-Ausfall problematisch für latenzsensitive Anwendungen
    - Metadatenverwaltung für viele kleine Dateien → **Master als Flaschenhals**
- 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.

Colossus

- Colossus File System (CFS): **Google-interner 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.