

Datenverwaltung in der Cloud

Motivation

Google File System

Windows Azure Storage: Stream Layer

Zusammenfassung



■ 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 Symposium on Principles of Distributed Computing (PODC '00), page 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

■ Literatur



Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

The Google file system

Proceedings of the 19th Symposium on Operating Systems Principles (SOSP '03), pages 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
 - Schreibenfragen hängen in der Regel große Datenmengen 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 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 (→ *Chunk-Server*)
 - * Zugriffsberechtigungen von Clients
 - Überwachung von *Chunk-Servern* mittels *HeartBeat*-Nachrichten
 - Koordinierung der Lastverteilung



Schreiboperationen

- Vorbereitungen beim Anlegen eines Datenblocks

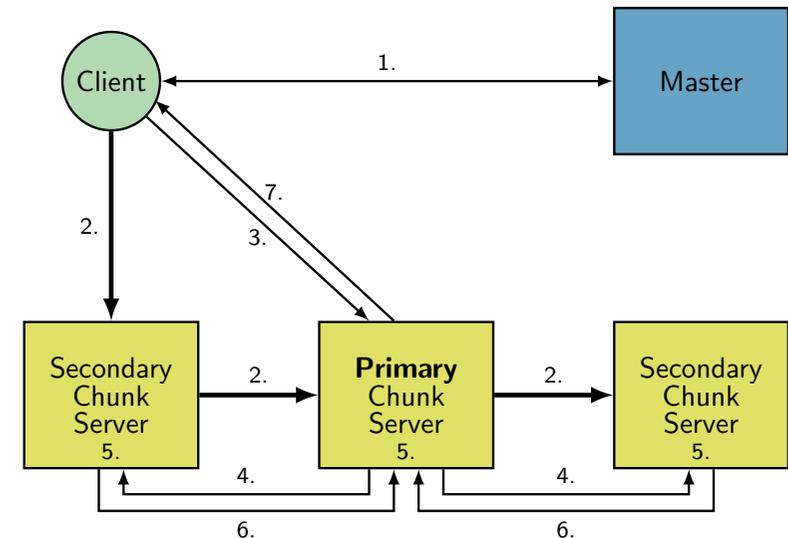
- Master wählt drei für den Datenblock zuständige *Chunk-Server* aus
- Einer der Server wird vom Master per Lease zum *Primary* ernannt, alle anderen Server übernehmen die Rolle von *Secondaries*

- Vorgehensweise bei Schreib Anfragen 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* leitet Schreibkommando an *Secondaries* weiter
5. *Chunk-Server* führen Schreiboperation aus
6. *Primary* sammelt Bestätigungen aller zuständigen *Chunk-Server*
7. *Primary* sendet Erfolgsmeldung an Client



Schreiboperationen



- **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 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



- **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



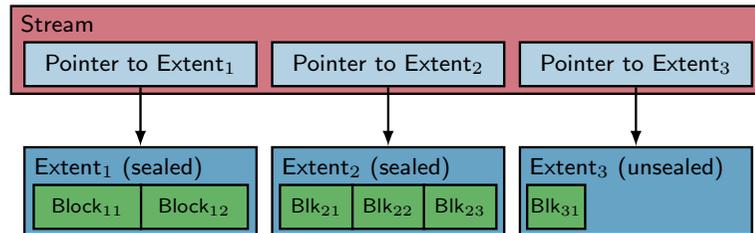
- **Fehler bei der Bearbeitung von Anhängoperationen**
[Vergleichbare Probleme können auch beim wahlfreien Schreiben auftreten.]
 - Mindestens ein Chunk-Server sendet keine Erfolgsbestätigung
 - Client erhält eine Fehlermeldung→ Client wiederholt die komplette Anhängoperation
- **Abgeschwächte Konsistenzeigenschaften**
 - Erfolgreiche Bestätigung einer Anhängoperation: Garantie, dass der Datensatz auf den Chunk-Servern am selben Offset gespeichert wurde
 - Potentielle Auswirkungen von Fehlern
 - 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
 - Berechnung von Checksummen durch die Anwendung
 - Einsatz von selbstverifizierenden, selbstidentifizierenden Datensätzen



- **Storage Stamp**
 - **Front-End Layer:** Empfang von Client-Anfragen
 - **Partition Layer:** Verwaltung von Blobs, Tabellen und Warteschlangen
 - **Stream Layer:** Direkter Zugriff auf Festplatten
- **Replikation auf zwei Ebenen**
 - **Innerhalb eines Stamp**
 - Aufgabe des Stream Layer
 - Synchroner Replikation während des Schreibvorgangs
 - **Zwischen Stamps**
 - Aufgabenteilung zwischen Partition Layer und Ortsdienst
 - Asynchrone Replikation im Hintergrund
- **Literatur**
 -  Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan et al. **Windows Azure Storage: A highly available cloud storage service with strong consistency.** *Proceedings of the 23rd Symposium on Operating Systems Principles (SOSP '11)*, pages 143–157, 2011.

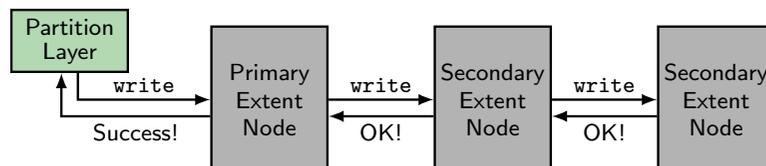


- **Block**
 - Kleinste Dateneinheit für Lese- und Schreibaufufe (variable Größe)
 - Periodische Überprüfung der Datenintegrität mittels Checksummen
- **Extent**
 - Datei (NTFS) mit aufeinander folgenden Blöcken
 - Zustände
 - Unversiegelt (*unsealed*): **Anhängen** weiterer Blöcke möglich
 - Versiegelt (*sealed*): Nur noch lesender Zugriff erlaubt
- **Stream**
 - Liste von Referenzen auf Extents
 - Nur der letzte Extent eines Stream ist unversiegelt



- **Extent Nodes**
 - Datenspeicherknoten
 - Aufgaben
 - Speicherung von Extents und ihrer Checksummen
 - Abbildung von Extent-Offsets zu Blöcken
 - Mehrere Festplatten pro Rechner
- **Stream Manager**
 - Verwaltungsknoten
 - Aufgaben
 - Erzeugung von Extents und Zuordnung zu Extent Nodes
 - Überwachung der Extent Nodes
 - Extent-Replikation zur Kompensation nach Hardware-Ausfällen
 - Garbage-Collection für nicht mehr referenzierte Extents
 - Verwaltung von Stream- und Extent-Informationen im Hauptspeicher
 - Stream Manager selbst ist ebenfalls repliziert

- **Anlegen eines neuen Extent**
 - Partition Layer weist Stream Manager an, einen neuen Extent zu erstellen
 - Stream Manager wählt drei Extent Nodes (einen *Primary*- und zwei *Secondary*-Knoten) aus verschiedenen Fehlerdomänen aus
 - **Hinzufügen eines Blocks zu einem Extent**
 - Partition Layer sendet Block an Primary Extent Node
 - Primary Extent Node zuständig für Koordinierung des Schreibaufrufs
 - Auswahl des Offset im Extent
 - Weiterleitung der Anfrage an die Secondary Extent Nodes
 - Primary Extent Node sendet Erfolgsbestätigung an Partition Layer
- Schreiben eines Blocks erfolgt ohne Einbeziehung des Stream Manager



- **Fehlersituationen (Beispiele)**
 - Fehlermeldung, dass ein Extent Node nicht erreichbar war
 - Fehlende Erfolgsbestätigung innerhalb einer vordefinierten Zeitspanne
- Partition Layer kontaktiert Stream Manager
- **Ausnahmebedingtes Versiegeln des aktuellen Extent**
 - Stream Manager befragt Extent Nodes nach aktuellem Extent-Offset
 - Versiegelung des Extent am kleinsten genannten Offset
 - **Anlegen eines (Ersatz-)Extent**
 - Auswahl einer neuen Gruppe von Extent Nodes
 - Wiederholung der Schreiboperation
 - **Anmerkungen**
 - Alle als „erfolgreich hinzugefügt“ bestätigten Daten bleiben erhalten
 - Ein einmal geschriebener Block wird u. U. mehrmals gespeichert
- Partition Layer muss damit umgehen können

■ Optimierung von Schreibzugriffen

■ Problem

- Intra-Stamp-Replikation erfolgt synchron → direkter Einfluss auf Antwortzeit
- Primary Extent Node muss auf Bestätigungen von Secondaries warten
- Bestätigung kann erst erfolgen, wenn der Block persistent gesichert wurde
- Instabile Antwortzeiten in Überlastsituationen („hiccups“)

■ Lösung

- Einsatz einer zusätzlichen Festplatte (*Journal Drive*)
- Doppelte Ausführung jeder Schreiboperationen: Journal Drive + Daten-Disk
- Senden der Bestätigung sobald einer der beiden Aufrufe erfolgreich war

■ Lastbalancierung für Leseanfragen

- Festlegung einer zeitlichen Schranke für die Bearbeitung einer Anfrage
- Senden der Anfrage an einen für den Block zuständigen Extent Node
- Extent Node schätzt ab, ob sich die zeitlichen Schranke einhalten lässt
 - Falls ja: Bearbeitung der Anfrage
 - Falls nein: Sofortige Ablehnung der Anfrage
- Bei Ablehnung: Neuer Versuch bei anderem Extent Node



■ 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
 - Abweichendes Verhalten bei Fehlern im Vergleich zu gewöhnlichen Dateisystemen möglich (z. B. einmalig geschriebener Datensatz)
 - Spezielle Vorkehrungen auf Anwendungsebene erforderlich

■ Windows Azure Storage Stream Layer

- Beschränkung auf eine Art von Schreiboperation: Anhängen von Daten
- Replikation auf verschiedene Fehlerdomänen
- Ähnliche Konsistenzgarantien wie beim Google File System
- Optimierungen zur Kompensation von Lastschwankungen

