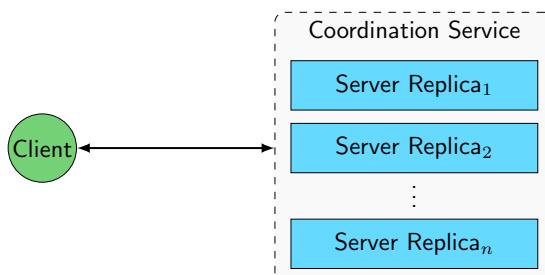


Koordinierungsdienste

- Motivation
- Grundkonzept
- Chubby
- ZooKeeper

Koordinierungsdienste

- Speicherung kleiner Datenmengen (< 1 MB)**
 - Verwaltung mittels Knoten in einer Baumstruktur
 - Dateisystemartiger Zugriff auf Nutz- und Metadaten
- Bereitstellung eines Benachrichtigungsdiensts**
 - Clients spezifizieren Ereignisse, über die sie informiert werden möchten
 - Beispielereignisse: Anlegen/Löschen eines Knotens, Änderung von Daten
- Fehlertoleranz durch **Replikation**



Koordinierungsaufgaben in der Cloud

- Charakteristika großer verteilter Cloud-Systeme und -Anwendungen
 - Hohe Anzahl von Komponenten bzw. Prozessen
 - Bereitstellung eines **gemeinsamen Diensts durch Kooperation**
- Problem: Kooperation erfordert **Mechanismen zur Koordinierung**
 - Beispiele für Koordinierungsaufgaben
 - Verteilter gegenseitiger Ausschluss
 - Wahl eines Anführers
 - Verteilte (Prioritäts-)Warteschlangen
 - Anforderungen
 - Zuverlässigkeit
 - Verfügbarkeit
- Zur eigentlichen Anwendung orthogonale Problemstellungen
- Herausforderungen
 - Wie lässt sich Koordinierung mittels eines **externen Diensts** bereitstellen?
 - Wie sehen mögliche Kompromisse zwischen Effizienz und Konsistenz aus?

Schnittstelle

Standardoperationen

Operation	Beschreibung
create()	Anlegen eines Knotens
getData()	Auslesen der Nutz- und Metadaten eines Knotens
setData()	Aktualisieren der Nutzdaten eines Knotens
delete()	Löschen eines Knotens

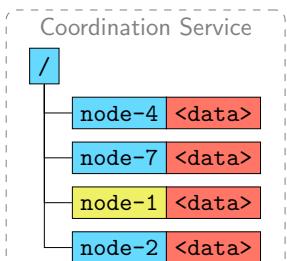
Implementierungsspezifisch: Zusätzliche Hilfsoperationen

getChildren()	Ausgabe der Pfadnamen von Kindknoten
setACL()	Aktualisierung der Zugriffsrechte für einen Knoten
[...]	

Bedingte Ausführung von Operationen

- Problem: Korrektheit vom vorherigen Zustand abhängiger Aktualisierungen
- Lösungsansatz
 - Verwaltung eines Versionszählers für jeden Knoten
 - Bearbeitung einer Operation nur, falls Versionsnummer weiterhin aktuell ist

Anwendungsbeispiel: Prioritätswarteschlange



Client

```
CoordinationService cs = create connection;

void insert(Object o, Priority p) {
    /* Encode priority in node name. */
    String nodeName = "/node-" + p;
    /* Create node and set its data to o. */
    cs.create(nodeName, o);
}

Object remove() {
    /* Find node with the highest priority.*/
    String[] nodes = cs.getChildren("/");
    String head = node from nodes with highest priority
    according to its name;

    /* Get node data and remove node. */
    Object o = cs.getData(head);
    cs.delete(head);
    return o;
}
```

Chubby

- Anforderungen
 - Dienst zur **Wahl eines Anführers**
 - Beispiel: Master-Server im Google File System
 - Typisches Nutzungsprofil eines Lock
 - Lock wird einmal angefordert
 - Wechsel des Lock-Halters üblicherweise nur nach Ausfällen

Chubby

- Ausrichtung auf **großgranulare Locks** [→ Haltezeiten von Stunden oder sogar Tagen]
- Bereitstellung starker Konsistenzgarantien
- Unterstützung einer großen Anzahl von Client-Prozessen [z. B. 90.000 [Burrows]]
- **Zuverlässigkeit wichtiger als Performanz**

Literatur

- Mike Burrows
The Chubby lock service for loosely-coupled distributed systems
Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI '06), S. 335–350, 2006.

Knoten

Kategorien

Persistente Knoten

- Existenz des Knotens unabhängig von Client-Sitzung
- **Explizites Löschen** durch einen Client erforderlich

Flüchtige Knoten (Ephemeral Nodes)

- Existenz des Knotens ist an eine Client-Sitzung gebunden
- **Automatisches Löschen** des Knotens bei Sitzungsende bzw. -abbruch
- Anwendungsbeispiel: Ausfallerkennung
 - * Client C erstellt flüchtigen Knoten k
 - * Andere Clients registrieren sich für Benachrichtigung über Löschung von k
 - * Ausfall von Client C → Koordinierungsdienst löscht Knoten k
 - * Registrierte Clients erhalten Mitteilung über Ausfall von Client C

Dienstunterstützte Knotennamen (Sequenzielle Knoten) [Nur in ZooKeeper.]

- Automatisch generierte **Sequenznummer als Teil des Knotennamens**
- Client spezifiziert beim Anlegen des Knotens den Namenspräfix
- Anwendungsbeispiel: Anführerwahl [Siehe Tafelübung.]



Architektur

Client-Bibliothek

- Senden von Anfragen an Chubby
- Verwaltung eines **Antwort-Cache für Nutz- und Metadaten**

Server-Seite: Chubby-Zelle

- Typische Größe
 - 5 Replikate
 - Tolerierung von 2 Ausfällen möglich
- Master-Replikat
 - **Ausführung aller (lesenden und schreibenden) Client-Anfragen**
 - Erzeugung und Verteilung von Zustandsaktualisierungen
 - Festlegung einer an das Master-Replikat gebundenen *Epochennummer*
- Andere Replikate
 - **Einspielen von Zustandsaktualisierungen**
 - Keine Aktion bei lesenden Anfragen
 - Wahl eines neuen Master-Replikats bei Ausfall des vorherigen



Chubby-spezifische Schnittstellenerweiterungen

Zugriff auf einen Knoten erfolgt mittels **Handle**

- Vergleiche: Dateideskriptor in UNIX
- Operationen
 - Open(): Erzeugung eines Handle
 - * Festlegung des Zugriffsmodus (z. B. Schreiben, Lesen, Lock-Anforderung)
 - * Spezifizierung der abonnierten Ereignisse
 - Close(): Schließen eines Handle

Locks

- Verfügbare Modi
 - Exklusives Schreiber-Lock
 - Geteiltes Leser-Lock
- Lock-Vergabe unabhängig vom Zugriff auf die Nutz- und Metadaten des korrespondierenden Knotens (*Advisory Locks*)
- Operationen
 - Acquire(): Anfordern eines Lock
 - Release(): Rückgabe eines Lock

Caching auf Client-Seite

Problem: **Master-Replikat stellt Flaschenhals dar**

- Bearbeitung schreibender und lesender Anfragen
- Verwaltung von Sitzungen
- Entlastung des Master-Replikats erforderlich

Lösungsansatz: **Einsatz von Client-Caches** für Nutz- und Metadaten

Invalidierung von Cache-Inhalten

- Master-Replikat verwaltet eine Liste mit Informationen darüber, welche Daten jeder Client in seinem Cache zwischengespeichert haben könnte
- Bei jeder **modifizierenden Anfrage**
 1. Master-Replikat sendet Invalidierungen an alle betroffenen Clients
 2. Clients löschen die entsprechenden Daten aus ihrem Cache
 3. Clients bestätigen dem Master-Replikat die Invalidierung
 4. Master-Replikat führt Anfrage aus, sobald alle Bestätigungen betroffener Clients vorliegen bzw. deren Sitzungs-Leases ausgelaufen sind
- Realisierung: Ausnutzung des KeepAlive-Fernauftruf-Mechanismus

Verwaltung von Client-Sitzungen

Grundlegendes Konzept

- Sitzung: **Temporäre Garantie von Chubby an einen Client**
 - Vom Client erzeugte Knoten-Handles bleiben gültig
 - Vom Client gehaltene Locks werden nicht neu vergeben
- Festlegung der Dauer einer Sitzung mittels Lease

Verlängerung einer Sitzung

- Bei Sitzungsaufbau
 - Client initiiert KeepAlive-Fernauftruf zum Master-Replikat
 - Master-Replikat **blockiert** in dem Fernauftruf
- Vor Ablauf eines Sitzungs-Lease
 - Master-Replikat **deblockiert** KeepAlive-Fernauftruf
 - Antwort des Fernauftrufs enthält Dauer der Lease-Verlängerung [Standardwert: 12 s]
 - Client startet erneuten KeepAlive-Fernauftruf

Vorzeitige Rückkehr aus einem KeepAlive-Fernauftruf

- Benachrichtigung eines Clients über abonnierte Ereignisse
- Vorteil: Ansatz funktioniert auch für durch Firewalls geschützte Clients

Verhalten bei Fehlersituationen

Korrechter Client schafft es nicht, sein Sitzungs-Lease zu verlängern

- Möglicher Grund: **Überlastung bzw. Ausfall des Master-Replikats**
- Vorgehensweise auf Client-Seite
 1. Löschen und Deaktivieren des eigenen Cache
 2. Eintritt in eine Wartephase (**Grace Period**, Dauer: z. B. 45 s)
 3. Nach Ablauf der Wartephase
 - * Abbruch sämtlicher Aufrufe
 - * Fehlermeldung an die Anwendung
 4. Bei Wiederherstellung einer Verbindung zu einem Master-Replikat
 - * Verlängerung des Sitzungs-Lease
 - * Reaktivierung des eigenen Cache

Wechsel des Master-Replikats (z. B. bei Ausfall): Aktionen des neuen Master

1. Wahl einer neuen Epochenummer → Identifikation veralteter Anfragen
2. Rekonstruktion der Master-Datenstrukturen im Hauptspeicher
3. Senden einer Fail-Over-Benachrichtigung an alle bekannten Clients
4. **Warten auf Bestätigungen** der kontaktierten Clients
5. Wechsel in den Normalzustand

- **ZooKeeper** [Weitere Details in der Tafelübung.]
 - Apache-Projekt: <http://zookeeper.apache.org/>
 - Produktiveinsatz: Yahoo, Facebook, Twitter,...
 - Verwaltung des Zustands im Hauptspeicher
 - Periodische Sicherungspunkte auf Festplatte
- Hauptunterschiede zu Chubby
 - Keine direkte Unterstützung von Locks
 - Keine vom System verwalteten Client-Caches
 - **Abgeschwächte Konsistenzgarantien**
 - Auf allen Replikaten: Identische Ausführungsreihenfolge für Schreibanfragen
 - Alle Anfragen desselben Clients werden in FIFO-Reihenfolge bearbeitet

Literatur

-  Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed
ZooKeeper: Wait-free coordination for Internet-scale systems
Proceedings of the 2010 USENIX Annual Technical Conference (ATC '10), S. 145–158, 2010.

11–13

- Server-Seite: **ZooKeeper-Zelle**
 - Typische Größe: 5 Replikate
 - Rollenverteilung
 - Leader-Replikat
 - Follower-Replikate
- Client: Sitzungsaufbau zu **beliebigem Replikat**
 - Ziel: Verteilung des Kommunikationsaufwands auf alle Replikate
 - Client kommuniziert im Normalfall ausschließlich mit einem Replikat
- Bearbeitung zustandsmodifizierender Anfragen
 1. Client sendet Anfrage an das mit ihm verbundene Replikat R
 2. Replikat R leitet die Anfrage an das Leader-Replikat weiter
 3. **Leader bearbeitet die Anfrage** und erzeugt eine Zustandsaktualisierung
 4. Ausführung der Zustandsaktualisierung auf allen Replikaten
 5. Nach lokaler Aktualisierung: Replikat R sendet Antwort an den Client

© td MWCC (WS19/20) Koordinierungsdienste – ZooKeeper

11–14

Optimierung für lesende Anfragen

- **Bearbeitung lesender Anfragen**
 1. Client sendet Anfrage an das mit ihm verbundene Replikat R
 2. Replikat R führt Anfrage direkt aus
 3. Replikat R sendet Antwort an den Client
- **Mögliche Konsequenz: Lesen veralteter Zustände**
 - Leseanfragen können Schreibanfragen anderer Clients „überholen“
 - Beispiel: Client C_1 ist mit Replikat R_1 verbunden, Client C_2 mit Replikat R_2
 1. Client C_1 legt erfolgreich einen Knoten k an
 2. Client C_1 sendet eine Nachricht an Client C_2 , dass Knoten k angelegt wurde
 3. Client C_2 erhält beim Versuch auf k zuzugreifen eine Fehlermeldung von Replikat R_2 , dass der Knoten nicht existiert

→ Replikat R_2 hatte die Anfrage von Client C_1 noch nicht ausgeführt
 - ZooKeeper-spezifische Schnittstellenerweiterung: sync-Operation
 - Bearbeitung aller ausstehenden Schreibanfragen
 - **Langsame Leseanfrage**: Auf sync-Aufruf folgende Leseoperation

11–15