


ZooKeeper
Überblick
Konsistenzwahrung in ZooKeeper
Aufgabe 5



- Fehlertoleranter Koordinierungsdienst für verteilte Systeme
 - Teil des Hadoop-Projekts
 - Im Produktiveinsatz (z. B. bei Yahoo und Facebook (Cassandra))
- Verwaltung von Daten
 - Hierarchischer Namensraum: Knoten in einer Baumstruktur
 - Jeder Knoten über eindeutigen Pfad identifizierbar (Wurzelknoten: „/“)
 - Knoten können Nutzdaten aufnehmen
- Fehlertoleranz
 - Aktive Replikation des Diensts auf mehreren Rechnern
 - Replikatkonsistenz mittels Leader-Follower-Ansatz
- 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, 2010.



- Zentrale Operationen
 - `create` Erstellen eines Knotens
 - `exists` Überprüfung, ob ein Knoten existiert
 - `delete` Löschen eines Knotens
 - `setData` Setzen der Nutzdaten eines Knotens
 - `getData` Auslesen der Nutz- und Metadaten eines Knotens
 - `getChildren` Rückgabe der Pfade von Kindknoten eines Knotens
 - `sync` Warten auf die Bearbeitung aller vorherigen zustandsmodifizierenden Operationen (siehe später)
- Aufrufvarianten
 - Synchron
 - Asynchron
- ZooKeeper-API (Version 3.2.2)
 - <http://hadoop.apache.org/zookeeper/docs/r3.2.2/api/>



- Persistente Knoten (*Regular Nodes*)
 - Erzeugung durch den Client
 - Explizites Löschen durch den Client
- Flüchtige Knoten (*Ephemeral Nodes*)
 - Erzeugung durch den Client unter Angabe des EPHEMERAL-Flag
 - Löschen
 - Explizites Löschen durch den Client
 - Automatisches Löschen durch den Dienst, sobald die Verbindung zum Client, der diesen Knoten erstellt hat, beendet wird oder abbricht
 - Anwendungsbeispiel: Benachrichtigung über Knotenausfall
- Sequenzielle Knoten (*Sequential Nodes*)
 - Erzeugung durch den Client unter Angabe des SEQUENTIAL-Flag
 - Automatische Erweiterung des Knotennamens um eine vom System vergebene Sequenznummer
 - Anwendungsbeispiel: Herstellung einer Ordnung auf Clients

[Hinweis: Das EPHEMERAL- und das SEQUENTIAL-Flag sind miteinander kombinierbar]



Verwaltung von Metadaten

- **Verwaltete Metadaten eines Knotens**
 - Zeitstempel der Erstellung
 - Zeitstempel der letzten Modifikation
 - Versionsnummer der Nutzdaten
 - Größe der Nutzdaten
 - Anzahl der Kindknoten
 - Bei flüchtigen Knoten: ID der Verbindung des ZooKeeper-Clients, der den Knoten erstellt hat (*Ephemeral Owner*)
 - ...
- **Kapselung der Metadaten eines Knotens in einem Objekt** der Klasse `org.apache.zookeeper.data.Stat`
- **Implementierungsentscheidung**
 - Nutz- und Metadaten werden komplett im Hauptspeicher gehalten
 - Keine Strategie für den Fall, dass der Hauptspeicher voll ist



Verwaltung von Nutzdaten

- **Grundprinzipien** [→ Unterschiede zu Dateisystemen]
 - Jeder Knoten kann Nutzdaten aufnehmen
 - Kleine Datenmengen, üblicherweise < 1 KB pro Knoten
 - „Verzeichnisknoten“ (also Knoten mit Kindknoten) können ebenfalls Nutzdaten direkt aufnehmen
 - Daten werden atomar geschrieben und gelesen
 - {S,Ers}etzen der kompletten Nutzdaten eines Knotens beim Schreiben
 - Kein partielles Lesen der Nutzdaten
- **Versionierung der Nutzdaten**
 - Schreiben neuer Daten → Inkrementierung der Knoten-Versionsnummer
 - Bedingtes Schreiben von Nutzdaten

```
public Stat setData(String path, byte[] data, int version);
```

 - Nutzdaten `data` werden nur geschrieben, falls die aktuelle Versionsnummer des Knotens `version` entspricht („test and set“)
 - Schreiben ohne Randbedingung: `version = -1` setzen
 - Kein Zugriff auf ältere Versionen möglich



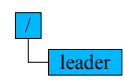
Benachrichtigung über Ereignisse

- **Problemstellung**
 - Client wartet darauf, dass ein bestimmtes Ereignis eintritt
 - Aktives Nachfragen durch den Client ist im Allgemeinen nicht effizient
- **Wächter (*Watches*)**
 - Objekt auf Client-Seite, das einen Rückruf (*Callback*) des ZooKeeper-Diensts entgegen nimmt
 - Anwendungsbeispiele: Benachrichtigung über
 - Erstellen eines Knotens
 - Löschen eines Knotens
 - Änderung der Nutzdaten eines Knotens
- **Schnittstelle für Wächter-Objekte**

```
public interface Watcher {
    public void process(WatchedEvent event);
}
```



Anwendungsbeispiel: Wahl eines Anführers

- **Problemstellung**
 - In einer Gruppe von ZooKeeper-Clients soll ein Anführer gewählt werden
 - Bei Ausfall des Anführers muss ein neuer Anführer bestimmt werden
 - **Umsetzung**
 - Erstellen eines „Verzeichnisknotens“ `/leader` für die Gruppe
- 
- **Vorgehensweise beim Hinzukommen eines neuen Clients**
 - Erstellen eines flüchtigen Kindknotens `/leader/node-<Sequenznummer>`
 - Suche nach Kindknoten mit kleineren Sequenznummern
 - Existiert kein Kindknoten mit kleinerer Sequenznummer → Client ist *Leader*
 - Sonst: Client ist *Follower* → Setzen eines Watch auf den Kindknoten mit der nächstkleineren Sequenznummer
 - **Bei Knotenausfall**
 - Automatische Löschung des zugehörigen flüchtigen Knotens
 - Genau ein Client wird per Watch über den Ausfall benachrichtigt



Anwendungsbeispiel: Wahl eines Anführers

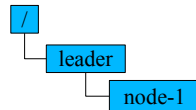
Beispielablauf

- Client 1 kommt neu zur Gruppe hinzu
 - Erstellen eines flüchtigen Kindknotens `/leader/node-1`
 - Client 1 wird zum Leader, da sein Kindknoten die kleinste Sequenznummer aufweist (bzw. in diesem Fall keine weiteren Kindknoten vorhanden sind)

Clients

Leader
①

ZooKeeper-Dienst



Anwendungsbeispiel: Wahl eines Anführers

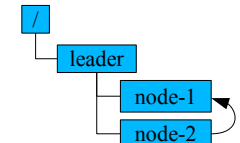
Beispielablauf

- Client 2 kommt neu zur Gruppe hinzu
 - Erstellen eines flüchtigen Kindknotens `/leader/node-2`
 - Client 2 wird zum Follower
 - Client 2 setzt Watch auf Kindknoten mit nächstkleinerer Sequenznummer (\rightarrow `/leader/node-1`)

Clients

Leader
①

ZooKeeper-Dienst



②



Anwendungsbeispiel: Wahl eines Anführers

Beispielablauf

- Client 3 kommt neu zur Gruppe hinzu
 - Erstellen eines flüchtigen Kindknotens `/leader/node-3`
 - Client 3 wird zum Follower
 - Client 3 setzt Watch auf Kindknoten mit nächstkleinerer Sequenznummer (\rightarrow `/leader/node-2`)

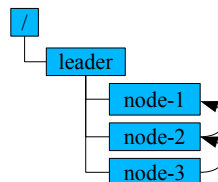
Clients

Leader
①

③

②

ZooKeeper-Dienst



Anwendungsbeispiel: Wahl eines Anführers

Beispielablauf

- Ausfall des Leader-Knotens Client 1
 - Abbruch der Verbindung zum ZooKeeper-Dienst
 - Automatische Löschung des Kindknotens `/leader/node-1`
 - Client 2 wird per Watch über den Ausfall benachrichtigt und steigt damit zum neuen Leader auf

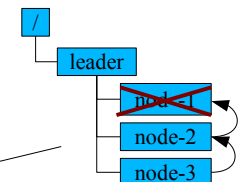
Clients

~~Leader
①~~

③

Leader
②

ZooKeeper-Dienst



ZooKeeper

Überblick

Konsistenzwahrung in ZooKeeper

Aufgabe 5



■ Problemstellung

- Aktive Replikation einer zustandsbehafteten Anwendung
- Replikatzustände müssen konsistent gehalten werden
- Beispiel für inkonsistente Zustände zweier Replikate R_1 und R_2
 - Zwei Anfragen A_1 und A_2 , die einem Knoten `/node` neue Daten zuweisen

A_1 : `setData("/node", new byte[] { 47 }, -1);`

A_2 : `setData("/node", new byte[] { 48 }, -1);`

- Annahme: A_1 erreicht R_1 früher als A_2 , bei R_2 ist es umgekehrt

R_1	/node-Daten	R_2	/node-Daten
< init >	null	< init >	null
A_1	[47]	A_2	[48]
A_2	[48]	A_1	[47]

■ Sicherstellung der Replikatkonsistenz in ZooKeeper

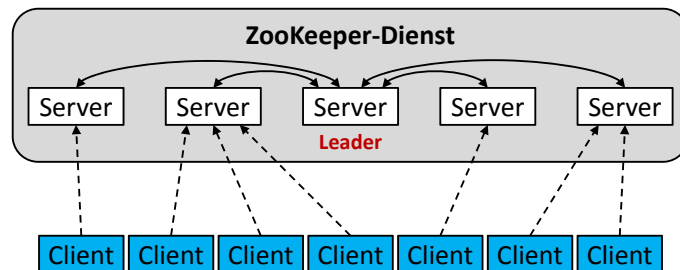
- Alle Replikate müssen Anfragen in der selben Reihenfolge bearbeiten
- Protokoll zur Erstellung einer Anfragenreihenfolge nötig



Replikation in ZooKeeper

■ Anzahl der ZooKeeper-Replikate

- Annahme: Ein ausgefallenes Replikat wird zeitnah durch ein neues ersetzt
- $2f + 1$ Replikate zur Tolerierung von höchstens f Fehlern bzw. Ausfällen
- Beispiel für $f = 2$



■ Leader-Follower-Ansatz

- Leader gibt Reihenfolge vor
- Follower bearbeiten Anfragen in der vorgegebenen Reihenfolge



Optimierung für lesende Anfragen

■ Erkenntnis

Lesende Anfragen haben auf die Replikatkonsistenz keinen Einfluss

■ Optimierung

- Lesende Anfragen werden nicht auf allen Replikaten ausgeführt, sondern nur auf dem Replikat, mit dem der Client direkt verbunden ist
- Bearbeitung einer lesenden Anfrage erfolgt sofort nach ihrem Empfang, auch wenn sie dadurch andere Anfragen „überholt“
- Vorteile
 - Einsparung von Ressourcen
 - Kürzere Antwortzeiten


■ Konsequenzen

- Die Antwort auf eine lesende Anfrage kann unterschiedlich ausfallen, je nachdem an welches Replikat die Anfrage geschickt wird
- Rückgabe von veralteten Daten und Versionsnummern möglich
- Erzwingen eines Synchronisationspunkts mit `sync()`
 - Warten bis alle vor dem `sync()` empfangenen Anfragen bearbeitet wurden
 - Anschließend Bearbeitung der lesenden Anfrage



Zab

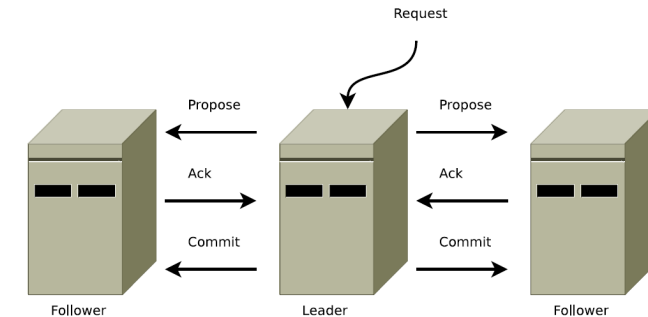
- Von ZooKeeper verwendetes Protokoll zur Reihenfolgeherstellung
 - In der Implementierung von Apache ZooKeeper nicht modular integriert
 - Nachträgliche eigenständige Implementierung als *Zab*
- *Totally Ordered Broadcast Protocol*
 - Leader-Follower-Ansatz
 - Zwei Protokoll-Modi
 - *Broadcast* Normalbetrieb
 - *Recovery* Wahl eines neuen Leader
- Source-Code
<https://svn.cs.hmc.edu/svn/linkedin08/zab-multibranch/src/java/main/>
- Literatur

 Benjamin Reed and Flavio P. Junqueira
A simple totally ordered broadcast protocol
Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware, pages 1-6, 2008.



Zab: Broadcast-Modus

- Ziel
Herstellung einer einheitlichen Reihenfolge aller Client-Anfragen
- Vorgehensweise
 - PROPOSE Leader schlägt Sequenznummer (zxid) für Anfrage vor
 - ACK Follower akzeptieren den Vorschlag
 - COMMIT Leader bestätigt die Sequenznummer der Anfrage



[Abbildung aus Reed et al. „A simple totally ordered broadcast protocol“]



Zab: Recovery-Modus

- Abbruch des Broadcast-Modus
 - Ausfall des Leader
 - Leader hat keine Mehrheit mehr
- Eigenschaften des Recovery-Protokolls
 - Eine Anfrage, die auf einem Knoten bestätigt wurde (COMMIT), wird (gegebenenfalls nachträglich) auf allen Knoten bestätigt
 - Nichtbestätigte Vorschläge werden verworfen
- Wahl eines neuen Leader
 - Ziel: Neuer Leader wird der Knoten, dem der Vorschlag mit der höchsten zxid bekannt ist; bei Gleichstand entscheidet die höhere Knoten-ID
 - Rundenbasierte Abstimmung, in der jeder Knoten jedem anderen seinen aktuellen Kenntnisstand mitteilt
 - Bei Fehlern während der Wahl: Neustart des Vorgangs nach Timeout
- Nach erfolgreicher Wahl
 - Leader stellt verloren gegangenen Vorschläge und Bestätigungen bereit
 - Wiederaufnahme des Broadcast-Modus



Zab: Implementierung

- Repräsentation eines Zab-Knotens in der abstrakten Basisklasse *Zab*
- Varianten von Zab-Knoten
 - *SingleZab* Einzelner (lokaler) Knoten
 - *MultiZab* Knoten als Teil einer verteilten Gruppe von Knoten
- Methoden

```
public void startup();
public void shutdown();
public ZabTxnCookie propose(byte [] message);
public ZabTxnCookie sync();
```

- *startup()* Starten eines Zab-Knotens
- *shutdown()* Stoppen eines Zab-Knotens
- *propose()* Senden einer zu ordnenden Nachricht
- *sync()* Senden einer Sync-Anfrage
- Transaktions-Cookie (*ZabTxnCookie*)
 - Interne Verwaltungsinformationen über zu ordnende Nachrichten
 - Sequenznummer, Server-IDs,...



Zab: Nachrichtenempfang & Zustandstransfer

- Empfang geordneter Nachrichten über die Schnittstelle `ZabCallback`

- Methoden

```
public void deliver(ZabTxnCookie id, byte message []);
public void deliverSync(ZabTxnCookie id);
public void status(ZabStatus status, String leader);
public void getState(OutputStream os);
public void setState(InputStream is, ZabTxnCookie lastCommit);
```

- `deliver()` Zustellung der nächsten geordneten Nachricht
- `deliverSync()` Benachrichtigung über eine Sync-Anfrage
- `status()` Benachrichtigung über Änderungen des Knotenstatus
- `getState()` Auslesen des aktuellen Knotenzustands
- `setState()` Schreiben des aktuellen Knotenzustands

- Knotenstatus (`ZabStatus`)

- `LOOKING` Temporärer Zustand während der Anführerwahl
- `FOLLOWING` Zustand eines Follower-Knotens
- `LEADING` Zustand des Leader-Knotens



Überblick

ZooKeeper

Überblick

Konsistenzwahrung in ZooKeeper

Aufgabe 5



Zab: Konfiguration

- Übergabe eines `Properties`-Objekts an den `Zab`-Konstruktor

- Parameter

- `myid` ID des lokalen Knotens
- `peer<i>` Adresse des Knotens *i*
- `dataLogDir` Verzeichnis, in dem das Transaktions-Log abgelegt wird
- `dataSnapDir` Verzeichnis, in dem die Snapshots abgelegt werden
- ...

- Beispielkonfiguration eines `MultiZab`-Knotens (insgesamt 3 Knoten)

- Zusammenstellung der Konfiguration

```
Properties zabProperties = new Properties();
zabProperties.setProperty("myid", String.valueOf(1));
zabProperties.setProperty("peer1", "localhost:12345");
zabProperties.setProperty("peer2", "localhost:12346");
zabProperties.setProperty("peer3", "localhost:12347");
```

- Initialisierung eines `Zab`-Knotens

```
ZabCallback zabListener = [...];
Zab zabNode = new MultiZab(zabListener, zabProperties);
```



Aufgabe 5

- Umsetzung des ZooKeeper-Diensts nach dem Vorbild der ZooKeeper-Implementierung von Apache

- Teilaufgaben

- Implementierung als Client-Server-Anwendung
- Replikation unter Zuhilfenahme von Zab
- Unterstützung flüchtiger Knoten

- Vereinfachte Schnittstelle

```
public String create(String path, byte[] data, boolean ephem);
public void delete(String path, int version);
public MWStat setData(String path, byte[] data, int version);
public byte[] getData(String path, MWStat stat);
```

- Fokus der Übungsaufgabe

- Konsistente Replikation eines zustandsbehafteten Diensts
- Unterschiedliche Behandlung von schreibenden und lesenden Anfragen
- Konsistente Zeitstempel



Konsistente Zeitstempel

■ Problemstellung

- Beim Erstellen eines Knotens bzw. bei jedem Setzen von Nutzdaten soll der Last-Modified-Zeitstempel in dessen Metadaten aktualisiert werden
 - Lokale Uhren der Replikate können voneinander abweichen
- ⇒ Inkonsistente Zeitstempel bei trivialer Implementierung

■ Lösungsansatz

- Leader ordnet jeder Transaktion einen Zeitstempel zu, der seiner lokalen Uhrzeit zum Zeitpunkt des Empfangs der Anfrage entspricht
- Bei der Ausführung einer Transaktion wird deren Zeitstempel als *aktuelle Zeit* verwendet – unabhängig vom gegenwärtigen Stand der lokalen Uhr

■ Weitere (zu weit führende) Fragestellungen

- Wie kann sichergestellt werden, dass die verwendeten Zeitstempel auch dann monoton steigend sind, wenn ein Leader ausfällt und ein anderer Knoten seine Rolle übernimmt, dessen Uhr nach geht?
- Wie ist mit Situationen umzugehen, in denen mehrere unterschiedliche Zeitstempel pro Transaktion benötigt werden, zum Beispiel zur Bestimmung der Bearbeitungsdauer einer Anfrage?
- ...



Ausgabeparameter in Java

■ Problem

- Methode soll mehr als ein Objekt zurückgeben
- Nur ein „echter“ Rückgabewert möglich

■ Lösungsmöglichkeiten

- Einführung eines Hilfsobjekts, das mehrere Rückgabewerte kapselt
- Verwendung von *Ausgabeparametern*

■ Beispiel für Ausgabeparameter: ZooKeeper-Methode `getData()`

- Aufruf: Übergabe eines „leeren“ Parameters

```
MWZooKeeper zooKeeper = new MWZooKeeper([...]);
MWStat stat = new MWStat(); // Leeres Objekt
zooKeeper.getData("/example", stat);
System.out.println("Version: " + stat.getVersion());
```

■ Intern: Setzen von Attributen des Ausgabeparameters

```
public byte[] getData(String path, MWStat stat) {
    [...] // Bestimmung der angeforderten Daten
    stat.setVersion(currentVersion);
    [...] // Setzen weiterer Attribute und Daten-Rueckgabe
}
```



Serialisierung & Deserialisierung von Objekten

■ Serialisierung & Deserialisierung in Java

- Objekte müssen das Marker-Interface `Serializable` implementieren
- {S,Des}erialisierung mittels `Object{Out,In}putStream`-Klassen

■ Beispiel für Serialisierung

```
public byte[] serialize(Serializable obj) throws Exception {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    ObjectOutputStream oos = new ObjectOutputStream(baos);
    oos.writeObject(obj);
    oos.close();
    baos.close();
    return baos.toByteArray();
}
```

■ Hinweis zum Einsatz von Object-Streams in Verbindung mit Sockets

- Der Konstruktor des `ObjectInputStream` blockiert solange, bis auf der anderen Seite der Verbindung ein `ObjectOutputStream` geöffnet wird
- ⇒ Object-Streams auf beiden Seiten in unterschiedlicher Reihenfolge öffnen



Logging mit log4j

■ Zab verwendet intern die Logging-API *log4j*

- Konfiguration mittels einer Datei `log4j.properties`, die im Classpath der Java-Anwendung abgelegt sein muss
- Granularitätsstufen: OFF, ERROR, WARN, DEBUG, ALL, ...

■ Beispiele für log4j-Konfigurationen

- Ausgabe der Log-Meldungen auf der Konsole (Stufe: DEBUG)

```
log4j.rootLogger=DEBUG, CONSOLE
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
```

- Ausgabe der Log-Meldungen in der Datei `zab.log` (Stufe: INFO)

```
log4j.rootLogger=INFO, FILE
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=zab.log
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
```

