Introduction to Data Analytics Zookeeper

András Varga IBM Consulting

Bratislava, 2022

Contents

- **1** Introduction and Motivation
- 2 Architecture
- 3 Workflow
- 4 Leader Election in General
- 5 Leader Election in ZooKeeper
- 6 Using ZooKeeper

Introduction and Motivation

Motivation

In a distributed systems the following problems can be difficult to solve:

- Synchronization
- Maintaining configuration information

This can be even more problematic when taking into account race conditions

In the past usually all applications came with some sort of own solution, which complicated co-operation and maintenance

Introduction and Motivation

Introduction

Definition

A Distributed Coordination Service for Distributed Applications

"Because Coordinating Distributed Systems is a Zoo"



ZooKeeper can be used for

- maintaining configuration information
- naming
- providing distributed synchronization
- providing group services

Basics

- The coordination with each other is achieved through a shared hierarchal namespace (similar to a filesystem)
- ZooKeeper itself is intended to be replicated over a sets of hosts called an *ensemble*
- ZooKeeper transactions are ordered
- It runs in Java
- Highly-available and provides good performance

Guarantees

- Sequential Consistency
 - Client updates are applied in order
- Atomicity
- Single system image
 - Client sees the same view regardless of which server is connected to
- Reliability
 - If the update succeeds it persists
- Timelessness
 - The client view is up-to-date within a time-period

– Data Model

The Namespace

- There is a hierarchical namespace
- The basic blocks of the namespace are called *znodes*
- The whole namespace is stored in memory (for fast access)
- The namespace starts with the root element (denoted by /)
- And each znode is identified by its path starting at root containing path elements separated by slash
- Every znode has a unique parent znode whose path is a prefix of the znode with one less element

- Architecture

Data Model

Example of ZooKeeper Namespace



- Architecture

– Data Model

Properties of the Znodes

- The znodes are similar to files and directories in the normal filesystem
- A znode cannot be deleted if it has any children
- Every znode can have data associated with it (every znode is a file and directory at the same time)
- Znodes have limited amount of space for data (Unlike real files)
- Znodes store metadata alongside real data, such as ACL, version number, timestamps, etc. called as *Stat Structure*

└─ Data Model

Ephemeral ZNode

- Is a temporary znode
- These znodes exist only as long as the session creating them is active
- Once the session is closed all ephemeral nodes are deleted
- Therefore these znodes can not have children



Sequential Znode

- Can be persistent or ephemeral
- When creating a sequential znode ZooKeeper append a monotonically increasing counter to the end of path
- This counter is owned by the parent znode of the sequential znode
- The number is aligned to 10 digits by left padding of 0s

└─ Data Model

Example of Sequential Znode

- Creating znode "named property" in /app1/block1
- /app1/block1/property0000000001 is created
- The next sequential znode with the same name will be /app1/block1/property0000000002, etc.
- The actual counter (000000002) is stored in znode /app1/block1

Architecture

— Data Model

Read, Write and Versioning

- Read and write of a znode data is atomic (reading or re-writing every data from the znode)
- Each new data update increases the version number of the znode
- The access one has on the znode is defined by ACL for each node
- The following accesses can be defined: CREATE (a child node), READ, WRITE, DELETE (a child) and ADMIN

Architecture Architecture

The Ensemble

- ZooKeeper is deployed to a set of servers (machines, nodes) called as ensemble
- These servers know each other and maintain a local copy of the in-memory namespace and the transaction logs and snapshots in the persistent store
- One of the servers acts as a *leader*
- As long as the majority of the servers is available the ZooKeeper is available

Architecture Architecture

Connections from Clients

- Client connects to a single ZooKeeper server
- They maintain a TCP connection during the whole communication
- When the client first connects to the ZooKeeper service the server creates a session, which will be associated with a client and when the client needs to connect to different server the same session will be reestablished



Ensemble Example





Sessions

- A session is created when a client connects to the ZooKeeper service
- A session can be in several different states: Connecting, Connected, Close, etc.
- The client sends periodic heartbeats to keep the session alive
- Requests in a session are executed in FIFO order
- A client is provided with a list of IP:Port related to the servers in the ZooKeeper ensemble
- When a server fails for any reason the client will try to re-establish the connection to a different server in the list

└─ Time in ZooKeeper

Time in ZooKeeper I

 Zxid - Every change to the ZooKeeper state receives a unique stamp (ZooKeeper Transaction Id). This exposes the total ordering of all changes to ZooKeeper

■ if *zxid*1 < *zxid*2 then *zxid*1 happened before *zxid*2

- Version numbers Every change to a znode increases its version. There are 3 types of version:
 - version number of data changes
 - cversion number of child changes
 - aversion number of ACL changes

└─ Time in ZooKeeper

Time in ZooKeeper II

- Ticks Serves in a communication between the servers. Servers use ticks to define timing of events such as status uploads, different timeouts, etc.
 - This is an internal time
 - Minimum session timeout = 2 times the tick time
- Real Time Used only for putting timestamps to the stat structure during znode creation/modification

 Architecture Watches

Watches Motivation

- Usually the data read operations are one-time actions
- This creates a problem, when a client needs to wait for data changes
- A solution with trying to read data in an infinite cycle from client side is inefficient
- Watches are implemented by ZooKeeper for efficient resolution of this problem

- Architecture Watches

Watches

- Client queries to read data from ZooKeeper have an additional option: a watch parameter
- If a watch is false the query acts as usual
- If the watch is set true the server will send out a notification to the client, when the next tracked change occurs

Architecture Watches

Properties of Watches

One-time trigger

- Once the event occurs and the notification is sent the watch is removed, no more notifications shall be sent to the same client (unless the watch is re-enabled)
- Sent to a client
 - ZooKeeper provides an ordering guarantee: a client will never see a change for which it has set a watch until it first sees the watch event
- The data for which the watch was set
 - ZooKeeper "provides" two kinds of watches one for data change and one for child change, each function (create, setData, etc.) triggers the respective watch for the given node and its parent (if needed)

- Architecture └─ Watches

New Options of Watches

- Persistent Watches
- Recursive Watches
 - Sending data for each change of the znode and its (recursive) descendants

High Level Workflow

- Each node in the ensemble own a replicated database
- In addition the leader uses
 - a dedicated request processor to handle incoming write requests from the follower nodes
 - atomic broadcast to send changes from the leader to followers
- Write request is handled by the leader, propagates the request to all followers, if at least half of the ensemble accepts the change is accepted
- Read is performed locally no cluster interaction needed

Workflow Chart





Ensemble vs Qourum

- Ensemble is the array of all servers
- The ensemble needs *majority* to handle requests
- This majority is called *qourum*
- So essentially the qourum is the array of nodes handling requests

-Workflow Quorum

Options of Qourum

- majority qourum
- using weights
- hierarchy groups

Leader Election in General

Problem description:

Definition

It is a process to select a single designated coordinator for a distributed computation among several available processes/nodes.

The scope of ZooKeeper:

- Servers have IDs
- Topology is a complete graph

Simple Solution

- Processes may fail
- Messages are delivered reliably
- Bully algorithms
 - Asking for election
 - Responding to the election message
 - The node with lowest/biggest ID sends victory message

ZooKeeper Specifics

This section contains the implementation specific to ZooKeeper

- Atomic Broadcast Protocol
- Leader Election

Implementing Messaging

- FIFO channel is created between each servers
- Implemented using TCP as
 - Ordered delivery
 - No message after close
- Messages are timestamped
- Each proposal is marked by zxid = epoch + counter

−Leader Election in ZooKeeper └─Leader Election, ZAB

Electing a New Leader

Two possible algorithms implemented:

- LeaderElection
- FastLeaderElection
- Both provide:
 - The leader has seen the highest zxid of all the followers

• A quorum of servers have committed to following the leader The leader election also includes post election synchronization of servers Leader Election in ZooKeeper

Fast Leader Election

- It is a bully algorithm
- Which elects the leader with:
 - Most recent transaction history
 - Biggest ID of all such nodes
 - The majority of the qourum must elect this server
- As long as the leader is active no other server can be elected as a new leader

Leader Election in ZooKeeper

Post Election Synchronization

- The new leader establishes a new zxid
- By getting a new epoch and the highest transaction ID
- As a next thing it will give a NEW_LEADER proposal

−Leader Election in ZooKeeper └─Leader Election, ZAB

ZAB - Handling the NEW_LEADER Request

- A follower will ACK the NEW_LEADER proposal after it has synced with the leader
- A follower will only ACK a NEW_LEADER proposal with a given zxid from a single server
- A new leader will COMMIT the NEW_LEADER proposal when a quorum of followers have ACKed it
- A follower will commit any state it received from the leader when the NEW_LEADER proposal is COMMIT
- A new leader will not accept new proposals until the NEW_LEADER proposal has been COMMITED



Places for Configuration

- Installation is mostly a configuration
- Main configuration file: conf/zoo.cfg
- This file can be shared across the whole ensemble



Basic Configuration

- clientPort
 - the port clients listen into
- dataDir
 - holds snapshots of database and transaction logs
- tickTime
 - time in miliseconds
- and a lot more can be configured

Starting ZooKeeper by issuing bin/zkServer.sh start



Replicating ZooKeeper

- The basic configuration creates a stand-alone ZooKeeper node
- For High Availability more nodes are needed
- As ZooKeeper uses a voting system, where majority decides
- So an *odd* number of servers is favored
- This ensemble must be defined during the configuration



Defining an Ensemble

Additional configuration in the conf/zoo.cfg file:

- initLimit
 - Amount of *ticks* a node must connect into the leader
- syncLimit
 - Amount of how out-of-date a node can be compared to the leader
- server.X
 - X number of the server
 - One line of definition for each server in the ensemble
 - value: hostname:communication_port:leader_election_port
 - example: myHost:2888:3888

Resolving Standard Problems Using ZooKeeper

- Resolutions are purely client based
- No support is needed from zookeeper side
- ZooKeeper is asynchronous in nature
- But, it can be used to resolve some synchronous problems, such as locks
- All of this solutions are published in *recipes*

References I

- "Apache[™] Zookeeper Official Web Page."
- "Big Data University."
- "Tutorialspoint for ZooKeeper."