Name: Kunal Baweja
UNI: kb2896

Review: Zookeeper - Wait-free coordination for Internet Scale Systems

Introduction

The Zookeeper research paper describes ZooKeeper as a service for coordinating processes of distributed applications which aims to provide a simple and high performance kernel for building more complex coordination primitives at the client side. Zookeeper incorporates various elements from group messaging, shared registers, and distributed lock services in a replicated, centralized service.

Zookeeper Architecture

The ZooKeeper service consists of a group of servers that use replication to achieve high availability and performance, similar to many other distributed systems. Its high performance enables applications comprising a large number of processes to use such a coordination kernel to manage all aspects of coordination.

Zookeeper has a simple pipelined architecture that allows hundreds or thousands of requests outstanding while still achieving low latency. This pipeline guarantees FIFO order on client side to enable clients to submit operations asynchronously. With asynchronous operations, a client is able to have multiple outstanding operations at a time and hence low latency because it does not blocks at any time for response.

Zookeeper Service

The two main contributions of Zookeeper service are:

1. Coordination Kernel
2. Coordination recipes

ZooKeeper provides its clients an abstraction of a set of data nodes called znodes, organized according to a hierarchical namespace. The znodes in this hierarchy are represented as data objects that clients manipulate through the ZooKeeper client API.
Each client can create two types of znodes:

1. Regular: create and delete explicitly
2. Ephemeral: clients create znodes and either remove explicitly or system automatically deletes when the session that created the znode terminates.

Zookeeper Data Model and Client API

The data model of ZooKeeper is a file system with a simplified API and allows only full data reads and writes, or a key/value table with keys. Unlike normal files of a file system, znodes map to abstractions of the client application that created them and typically correspond to meta-data used for coordination purposes.
The Zookeeper Client API provides following functions:

1. Create - create a znode
2. Delete - deletes a znode
3. Exists - returns true if a znode exists
4. getData - fetches desired data and metadata
5. setData - writes data to znode
6. getChildren - returns znode children nodes of a given znode
7. Sync - Waits for all updates pending at the start of the operation to propagate to the server

Zookeeper Guarantees

Zookeeper guarantees **Linearizable writes** and **FIFO client order** i.e all requests that update the state of ZooKeeper are serializable and respect precedence and all requests from a given client are executed in the order that they were sent by the client, respectively.

Zookeeper Primitives

Zookeeper allows implementation of a range of primitives such as Configuration Management, Group Membership of ephemeral nodes, Simple Locks, Read/Write Locks etc.

Zookeeper Implementation

ZooKeeper provides high availability by replicating the data on each server that is present in the service. It further guarantees low latency as well.

Upon receiving a request, a server prepares it for execution called as request processor. If such a request requires coordination among the servers (write requests), then they use an agreement protocol (an implementation of atomic broadcast), and finally servers commit changes to the ZooKeeper database fully replicated across all servers. This database is maintained in-memory across all servers.

Every ZooKeeper server services clients and each client connects to only one server. As part of the agreement protocol write requests are forwarded to a single server, called the *leader*. The rest of the ZooKeeper servers, called followers, *receive* message proposals from other servers and agree via a consensus protocol on the state of the system.

During client server communication, when a server processes a write request, it also sends out and clears notifications corresponding to that update. As guaranteed by Zookeeper, servers process writes in order and do not process other writes or reads concurrently. Read requests are handled locally at each server. Each read request is processed and tagged with a zxid that corresponds to the last transaction seen by the server. This zxid defines the partial order of the read requests with respect to the write requests. This local processing provides enhanced performance at each server.