Distributed Systems
236351
Tutorial 5
ZooKeeper

Dolev Adas
(Revised by Yehonatan Buchnik)

Technion

November 22, 2018
Overview

Motivation

- Because Coordinating Distributed Systems is a Zoo
- Open source high-performance coordination service for distributed applications:
  - Naming
  - Configuration management
  - Synchronization
  - Group services
- http://zookeeper.apache.org/
Architecture

Overview

![ZooKeeper Service Diagram]

- ZooKeeper Service
- Leader
- Server
- Client
Architecture

Overview

- ZooKeeper service is replicated over a collection of pre-known servers called an Ensembles.
- As long as a majority of the servers are available, the ZooKeeper service will be available.
- Clients connect to a single ZooKeeper server and maintains a TCP connection through which it sends requests, gets responses, gets watch events, and sends heart beats. If the TCP connection to the server breaks, the client will connect to a different server.
- ZooKeeper stamps each update with a number that reflects the order of all ZooKeeper transactions.
Architecture

Ensembles

Zookeeper Service

- Delegation of write requests to Leader Node
- Leader Node
- Node 1: In-memory DB, Atomic Broadcast
- Node 2: In-memory DB, Atomic Broadcast
- Node 3: In-memory DB, Atomic Broadcast
- Node N: In-memory DB, Atomic Broadcast

Clients
Node components

Request Processor

The Request Processor component is only active in Leader Node and is responsible for processing write requests originating from client or follower nodes. Once request processor processes the write request, it **Atomic-Broadcast** the changes to follower nodes so that they can update their state accordingly.
Node components

Atomic Broadcast

The Atomic Broadcast component is present in both Leader Node and Follower Nodes. This component is responsible for broadcasting the changes to other nodes (in Leader Node) as well as receiving the change notification (in Follower Nodes). Atomic-Broadcasting two messages $m_1, m_2$ guarantees that they both Atomic-Delivered by all nodes in the same order.
Node components

In-memory Database

The in-memory database component responsible for storing the data in ZooKeeper. Every node contains its own database that enables them to server read requests. In addition to this, data is also written to file system providing recoverability in case of any problems with cluster. In case of write requests, inmemory database is updated only after it has successfully been written to file system.
Sessions

- Once a client connects to a server, the session will be established and a session id is assigned to the client.
- The client sends heartbeats at a particular time interval to keep the session valid. If the ZooKeeper ensemble does not receive heartbeats from a client for more than the period (session timeout) specified at the starting of the service, it decides that the client died.
- Session timeouts are usually represented in milliseconds.
- Requests in a session are executed in FIFO order.
Apache ZooKeeper stores its data in hierarchical Znodes (data nodes as in a tree structure) structure where each Znode can have its own data along with the child Znodes depending upon the type of Znode.

- A name is a sequence of path elements separated by a slash (/).
- Every Znode in ZooKeeper’s name space is identified by a path.
Data Model

Znodes

- Data stored into Znodes should not be more than a few KBs as it is supposed to be coordination data such as status information, configuration etc.
- Znodes maintain a status structure that includes version numbers for data changes, Access Control List (ACL) changes, and timestamps.
- The data stored at each Znode in a namespace is read and written atomically.
- Unlike a typical file system, which is designed for storage, ZooKeeper data is kept in-memory.
Znodes
Persistence Znodes

- Persistence Znode is alive even after the client, which created that particular Znode, is disconnected.
- By default, all Znodes are persistent unless otherwise specified.
- Use case: configuration data.
Znodes

Ephemeral Znodes

- Exists as long as the session that created the Znode is active.
- When the session ends the Znode is deleted.
- Are not allowed to have children.
- Use case: group membership.
Sequential Znodes can be either persistent or ephemeral.

When a new Znode is created as a sequential Znode, then ZooKeeper sets the path of the Znode by attaching a 10 digit sequence number to the original name. This counter is unique to the parent Znode.

Sequential Znodes play an important role in Locking and Synchronization.
Watchers
Sequential Znodes

- Clients can set a watcher on a Znode.
- A watcher will be triggered and removed when the Znode changes.
- When a watcher is triggered the client receives a message saying that the Znode has changed.
 Guarantees

- Sequential Consistency
- Atomicity
- Single System Image
- Reliability
- Timeliness
Reading and Writing

- Upon receiving a read request the zookeeper server gets the file from its own DB replica.
- Upon receiving a write request, the zookeeper server directs the request to the current leader. The leader in its turn, initiates the Atomic Broadcast protocol to disseminates the request.
- Reading is much faster than writing.
package omega;
import org.apache.zookeeper.*;
import java.io.IOException;
import java.util.Collections;
import java.util.List;

public class Omega implements Watcher {
    static ZooKeeper zk = null;
    static String root = "/OMEGA";
    static int ID;
    static int elected;
    Object lock = new Object();
    public Omega(String zkHost, int id) {
        try {
            zk = new ZooKeeper(zkHost, 3000, this);
            ID = id;
            elected = -1;
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
Example

Omega FD

...}

    public void propose() throws KeeperException, InterruptedException {
        if (zk.exists(root, true) == null) {
            zk.create(root, new byte[] {}, ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);
        }
        zk.create(root + "/", String.valueOf(ID).getBytes(), ZooDefs.Ids.OPEN_ACL_UNSAFE,
            CreateMode.EPHEMERAL_SEQUENTIAL);
    }

    public void electLeader() throws KeeperException, InterruptedException {
        synchronized (lock) {
            List<String> children = zk.getChildren(root, true);
            Collections.sort(children);
            byte[] data = null;
            for (String leader : children) {
                data = zk.getData(root + "/" + leader, true, null);
                if (data != null) {
                    break;
                }
            }
            if (data != null) {
                elected = Integer.parseInt(new String(data));
            }
        }
    }

...
...  

    public int getLeader() {
        synchronized (lock) {
            return elected;
        }
    }

    public void process(WatchedEvent watchedEvent) {
        final Event.EventType eventType = watchedEvent.getType();
        try {
            electLeader();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
import omega.Omega;
import static java.lang.String.format;

public class app {
    public static void main(String argv[]) {
        Omega omgea = new Omega("127.0.0.1", Integer.parseInt(argv[0]));
        try {
            omgea.propose();
            while (omgea.getLeader() == -1);
            System.out.println(format("leader is %d", omgea.getLeader()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
For Further Reading

- ZooKeeper official website
- source code