Zookeeper Notes

CSE 40822 – Cloud Computing

Caution: These are high level notes that I use to organize my lecture. You may find them useful for reviewing main points, but they aren't a substitute for participating in class.

References:

- Patrick Hunt, Mahadev Konar, Flavio Junqueira, and Benjamin Reed, Zookeeper: Wait-free coordination for Internet-scale systems, USENIX ATC 2010. https://www.usenix.org/legacy/event/usenix10/tech/full_papers/Hunt.pdf
- The Apache Zookeeper Project: http://zookeeper.apache.org

Need for a Coordination System

We have studied many different systems that offer weak consistency to the end user. (CAP Theorem) While the content of the system often has weak consistency, the internal configuration of the system often requires strict agreement on key items:

- What process is the leader/master of the group.
- What configuration file should be used by all processes.
- What processes are currently members of the group.
- What order should client requests be processed in?

In an operating system we would solve this with some basic synchronization primitives (critical section, mutex, monitor, etc) but these are hard (impossible?) to implement in the presence of failures.

Zookeeper is designed to be a single implementation of consistency primitives that many different applications can build upon to meet their needs.

Zookeeper Semantics

Architecture: A static set of servers comprises the ZK service. A client connects to the service via a session, and then issues multiple requests. If the client fails or is disconnected, both sides notice and may take action.

Semantics: ZK looks like a filessystem: it contains a hierarchy of znodes, each of which can store a small amount of data. Each znode can have an arbitrary number of children, each of which has a unique name (in that directory). Znodes can be accessed by complete path names /a/b/c/ or relative to a current working directory. Hard limit of 1MB data per node, but typically much smaller.

Fundamental Operations:

- create(path,data,flags) – path must be unique, flags may include ephemeral or sequential

- delete(path,version) – delete if znode matches version
- exists(path,watch) – return true if znode exists
- getData(path,watch) – get data associated with znode
- setData(path,data,version) – set data associated with znode, if version matches.
- getChildren(path,watch) – list children of this znode
- sync(path) – wait for all operations in session to propagate

Special flags:

- Sequential create: znode is created with an unique integer attached to the name, larger than any other in the directory.
- Ephemeral create: znode is automatically deleted when the client session ends
- Watch = Client will be notified when the znode is updated.

Example Applications

Configuration Management – Easy: put the config file in zookeeper, watch it.

Rendezvous – Workers look for master address in ZK, master publishes it when ready.

Group Membership – Each process creates an ephemeral/sequential znode in a shared directory.

Locking – Lock: Create ephemeral node. If it succeeds, you have the lock. If it exists, retry.

Unlock – Delete the emphemeral node. Problem: herd effect.

Improved lock:

```
N = create (dir+"/lock-", EPHEMERAL|SEQUENTIAL);
while(true) {
      C = getChildren(dir,false);
      If N is lowest znode in C, success so return
      If N is second lowest znode
            watch the lowest node b/c we are next.
}
```

Unlock:

```
Delete(N)
```

R/W locks – Write second procedure for read locks, which proceeds unless there is a write lock.

Barrier – Only proceed when N processes reach the same point.

Implementation

ZK consists of a static set of servers – all participants must know all server names.

The database is stored in memory and replicated across all nodes.

A transaction log is kept on disk for recovery from failures.

A leader process is elected, which coordinates write operations.

Writes are ordered and committed using an atomic broadcast protocol.

Reads are served from memory, but may be stale unless you sync first.

2F +1 servers are needed to tolerate up to F servers failed at once.

Summary

While many parts of a distributed system may be inconsistent (or weakly consistent), a few key properties must be agreed upon with strict consistency.

Zookeeper provides basic primitives which can be used to implement consistency operations for multiple applications simultaneously.

Zookeeper itself must have a static, globally known configuration in order to support dynamic configuration of other applications.