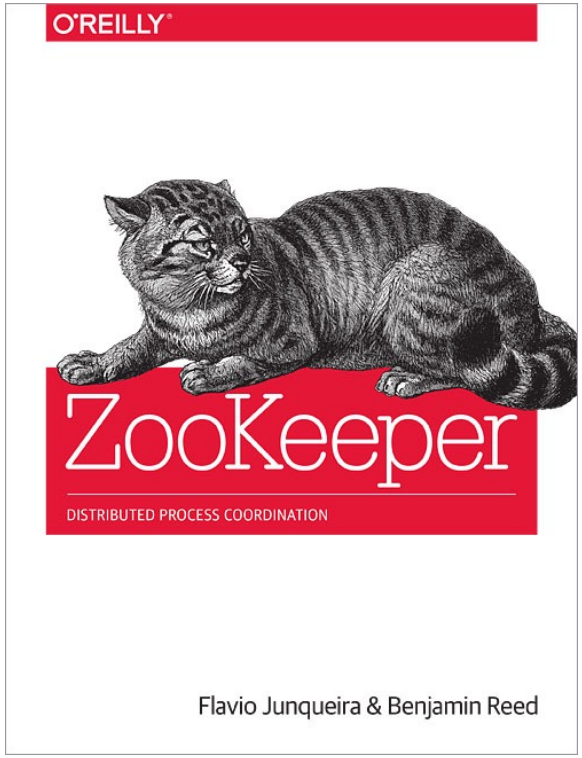
An Introduction to ZooKeeper (ZK)



Prof. Wing C. Lau
Department of Information Engineering
wclau@ie.cuhk.edu.hk

ZooKeeper – Distributed Process Coordination, by Flavio Junqueira, Benjamin Reed, Published by O'Reilly, Nov. 2013.

Acknowledgements

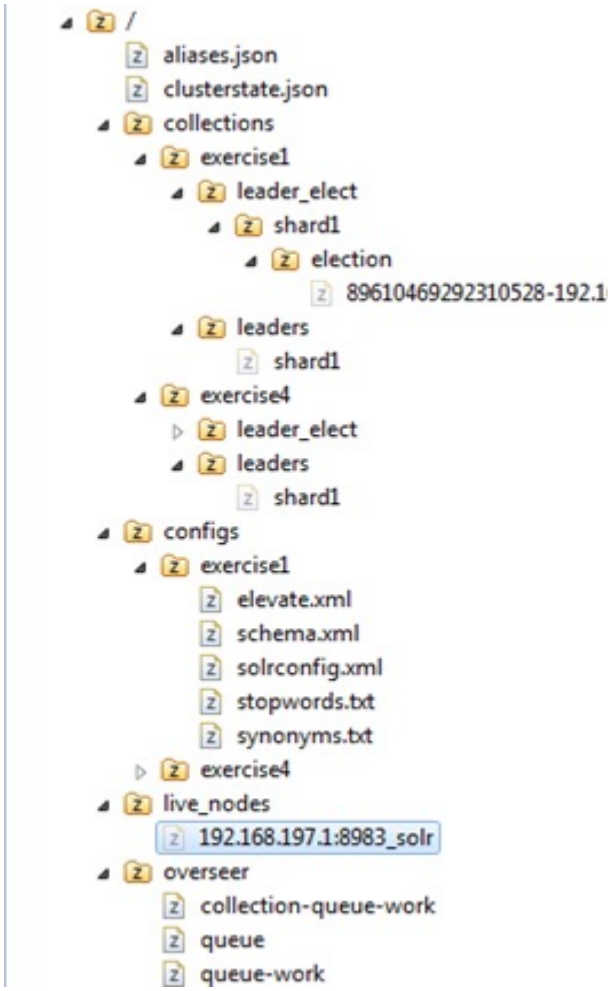
- The slides used in this chapter are adapted from the following sources:
  - Roy Campell, “Paxos and ZooKeeper,” Lecture notes of CS498 Cloud Computing, UIUC course, Spring 2014.

What is ZooKeeper?

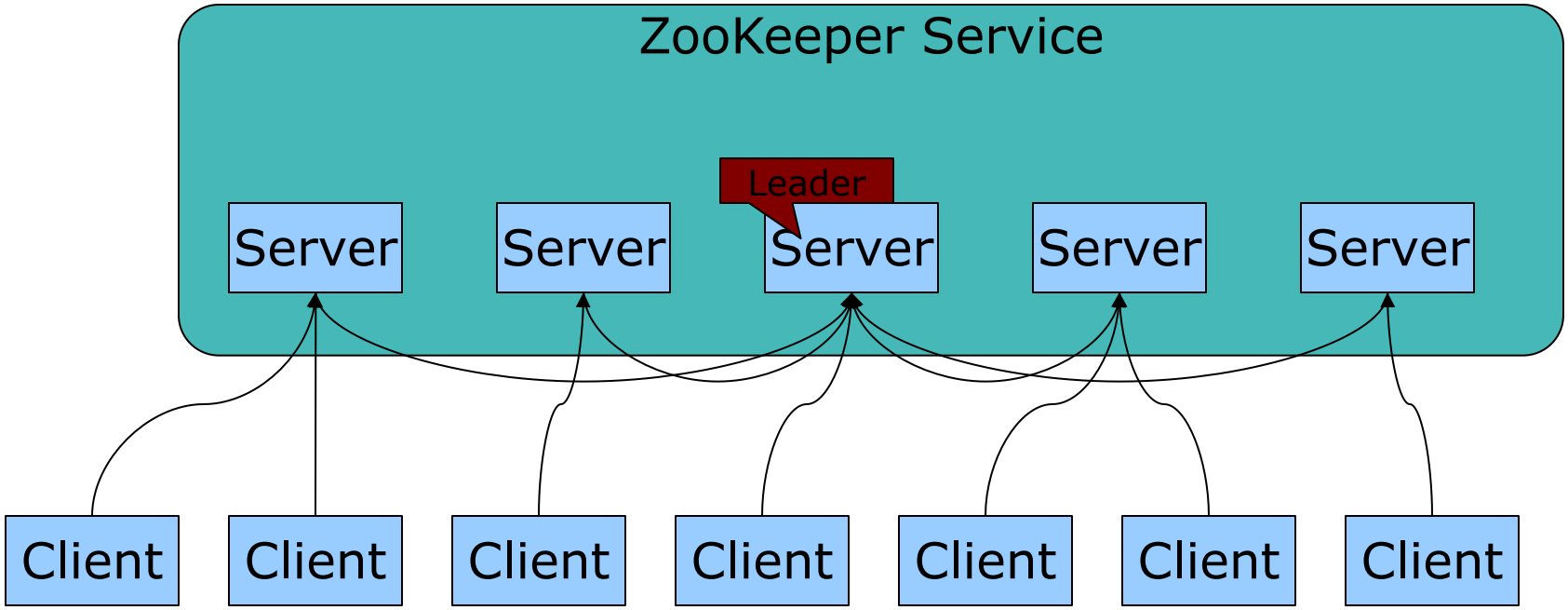
- A highly available, scalable service to support Distributed configuration, Consensus, Group Membership, Leader Election, Naming, and Coordination
- Difficult for Users to implement these kinds of services reliably
  - brittle in the presence of change
  - difficult to manage
  - different implementations lead to management complexity when the applications are deployed
- Originally developed by Yahoo! ; subsequently under Apache license: http://zookeeper.apache.org/

The Data Model of ZooKeeper (in form of a Special distributed filesystem tracked by ALL servers in a ZooKeeper service)

- Hierarchical Namespace
- Each node is called “znode”
- UNIX-like notation for path
- Each znode has data(stores data in byte[ ] array) and can have children
- Znode
  - Key Object for keeping/realizing the Consistent Distributed State info
  - Maintains “Stat” structure with version of data changes , ACL changes and timestamp
  - Version number increases with each change

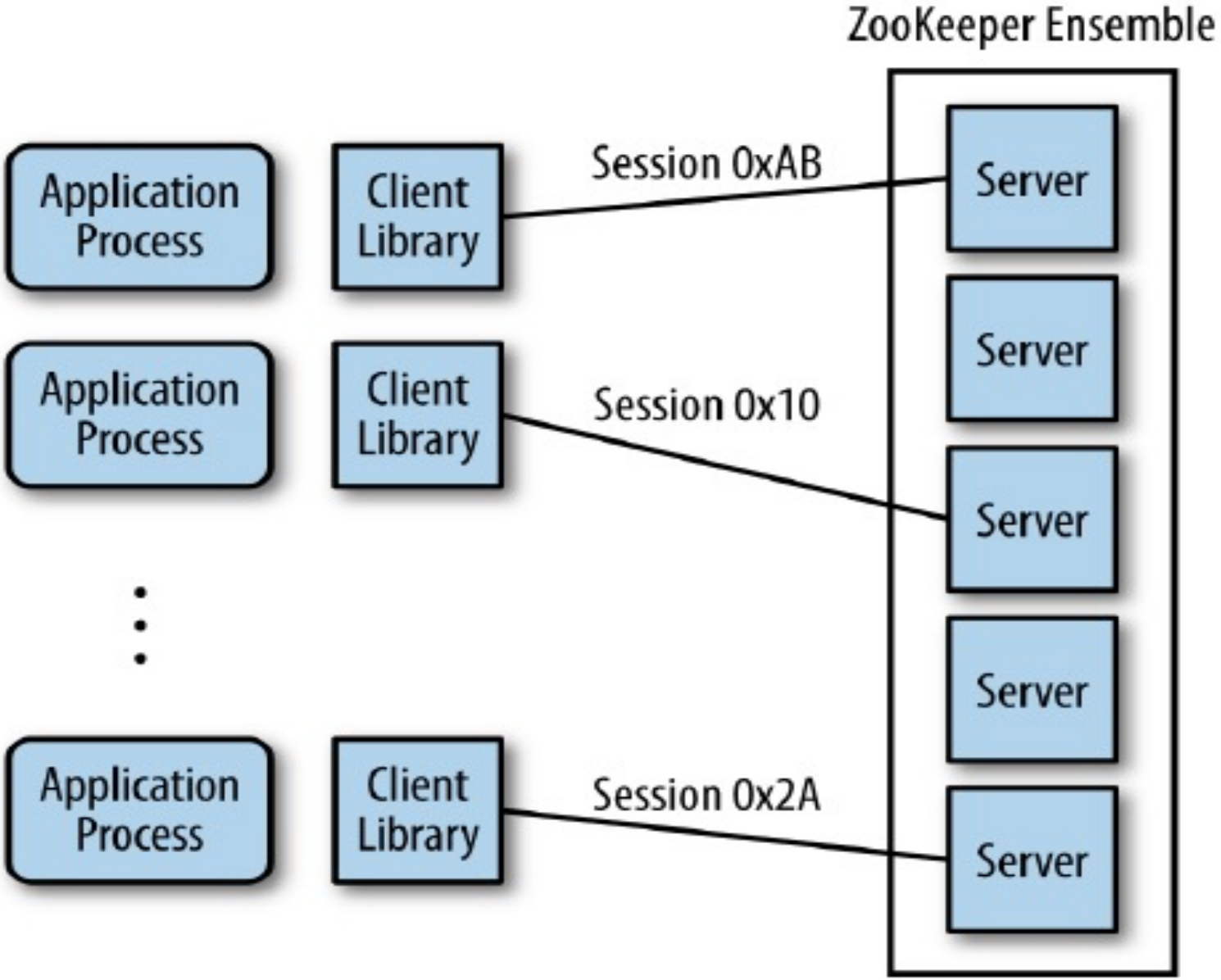


ZooKeeper Service



- All servers store a copy of the data, logs, snapshots on disk and use an in memory database
- A leader is elected at startup
- Followers service clients
- Update responses are sent when a majority of servers have persisted the change

How applications use the ZooKeeper services



Client API

Very similar to “standard” Filesystem access API:

- Create(path, data, flags)
- Delete(path, version)
- Exist(path, watch)
- getData(path, watch)
- setData(path, data, version)
- getChildren(path, watch)
- Sync(path)
- Two version synchronous and asynchronous

Zookeeper Service

- Watch Mechanism
  - Get notification
  - One time triggers
- Other properties of Znode
  - Znode is not designed for data storage, instead it stores meta-data or configuration
  - Can store information like timestamp version
- Session
  - A connection to server from client is a session
    - ZK also supports a Client to actively switch its connection to a different server while preserving the SAME Session
  - Bulit-in timeout mechanism
  - IMPORTANT: Ordering Guarantees by ZK are only valid for exchanges WITHIN the SAME Session ;
    - Special Care is needed to handle Broken (time-out) Sessions !

Properties Guaranteed by ZooKeeper

- Single System Image
  - The SAME client will see the same view of the service no matter which server it connects to ;
  - Different Clients may see a different (delayed) version of the view though ;
- Durability - once an update has been applied, it will persist from that time forward until a client overwrites the update.
- High Availability - 2F+1 servers can tolerate F crash failures
- Timeliness – The client's view of the system is guaranteed to be up-to-date within a certain bound delay.
  - By setting the "watch" flag in a Read request, a client will get notified of a change to data it is watching within a bounded period of time.
  - Either system changes will be seen by a client within this bound or the client will detect a service outage
  - There are corner cases that intermediate state-changes can be missed by a particular client due to the "one-time-trigger" notion of watch (see p.g. 70 of the ZK book)

Properties Guaranteed by ZooKeeper

- FIFO per Client Order
  - All requests of the SAME client will be applied/ processed in the order they were sent.
  - Ordering of notifications & state changes to/from a Client are guaranteed
- All Clients will observe "parallel" writes issued by different Clients (causing ZooKeeper to Change some global state) in the SAME order ;
  - Realized using the Zookeeper Atomic Broadcast (ZAB) protocol/ algorithm (http://research.yahoo.com/files/ladis08.pdf)
  - ZAB is inspired by, but different from, the **Paxos** algorithm
  - Reads issued by different clients on the same written variable can be different though (due to delay in state propagation to different ZK servers) !
- Atomicity - updates either Succeed or Fail, no partial results
  - File API without partial reads/writes
  - Simple **wait free** data objects organized hierarchically as in filesystem.
  - "Multi-hop" construct to support atomic (i.e. all or nothing) execution of a block of multiple commands/requests

Guarantees of the Zookeeper Atomic Broadcast (ZAB) Protocol run amongst ZK servers

- Replication Guarantees
  - Reliable Delivery – If a transaction, M, is committed by one server, it will eventually be committed by ALL servers.
  - Total Order – If Transaction A is committed before Transaction B by one server, A will be committed before B on ALL servers. If A and B are committed messages, either A will be committed before B or vice versa, i.e. A & B cannot be committed at exactly the same time !
  - Casual Order – If Transaction B is sent after Transaction A has been committed by the sender of B, A must be ordered before B. If a sender sends C after sending B, C must be ordered after B.
- Transactions are replicated as long as majority (quorum) of servers are up
- When servers fail and later restarted – it should catch up the transactions that were replicated during the time it was down.

Note: Under ZK, clients of the ZK services need to resync state with its ZK server once their session is broken (time-out)

Examples of ZooKeeper primitives

- Simple Lock
  - Create a znode L for locking
  - If one gets to create L he gets the lock
  - Others who fail to create watch L
  - Problems: herd effect

Examples of ZooKeeper primitives

- Simple Lock without herd effect

Lock

```
1 n = create(l + "/lock-", EPHEMERAL|SEQUENTIAL)
2 C = getChildren(l, false)
3 if n is lowest znode in C, exit
4 p = znode in C ordered just before n
5 if exists(p, true) wait for watch event
6 goto 2
```

Unlock

```
1 delete(n)
```

Examples of ZooKeeper primitives

- Leader Election
  - Any process which wants to be a leader will try to create (write) a znode, say /leader, with the EPHEMERAL flag ;
  - If multiple processes try at the same time, only the 1st one can complete the write successfully while all others will fail (because /leader node already exists as all writes are observed by the servers in the same order) ;
    - The other processes, after failing their write attempt, can still do a read on the /leader znode with the “watch” flag set.
    - When the current leader crashes or terminates its session explicitly, the znode /leader will disappear and all “watching” processes will get notified.

    => This scheme works but may have poor performance due to the so-called “herd” effects ! Why ?

Examples of ZooKeeper primitives

- Configuration Management
  - For dynamic configuration purpose
  - Simplest way is to make up a znode c for saving configuration.
  - Other processes set the watch flag on c
  - Due to the “One-time trigger” nature of ZK notification, a notification just indicates there is at least one update without telling how many time updates occurs

Examples of ZooKeeper primitives

- Rendezvous
  - Configuration of the system may not be sure at the beginning
  - Create a znode r for this problem
  - When master (leader) starts, it will fill the configuration in r
  - Workers watch node r
  - Set to ephemeral node

Examples of ZooKeeper primitives

- Group Membership
  - The Leader creates a znode g
  - Each group member process creates a znode for itself under g in ephemeral mode
  - Watch g for group membership changes

Examples of ZooKeeper primitives

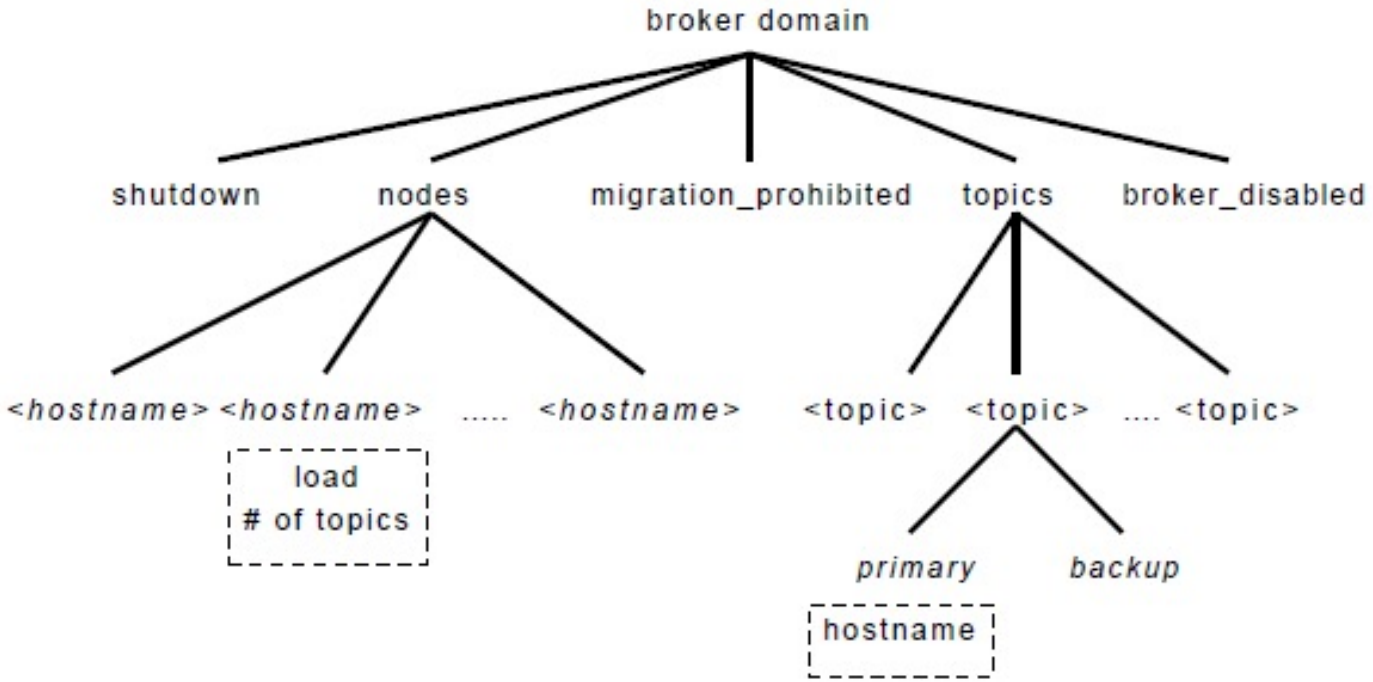
- Double Barrier
  - To synchronize the beginning and the end of computation
  - Create a znode B, and every process needs to register on it, by adding a znode under B
  - Set a threshold that start the process
  - Processes enter the barrier when # of child znodes exceeds the threshold (by watching on B)
  - When a process is done and ready to leave, it removes its child znode in B.
  - Processes can leave the barrier when every process has removed its child znode under B ; (by watching on B)

ZooKeeper Application Example

- Fetching Service
  - Using ZooKeeper for recovering from failure of masters
  - Configuration metadata and leader election

ZooKeeper Application Example

- Yahoo Message Broker
  - A distributed publish-subscribe system

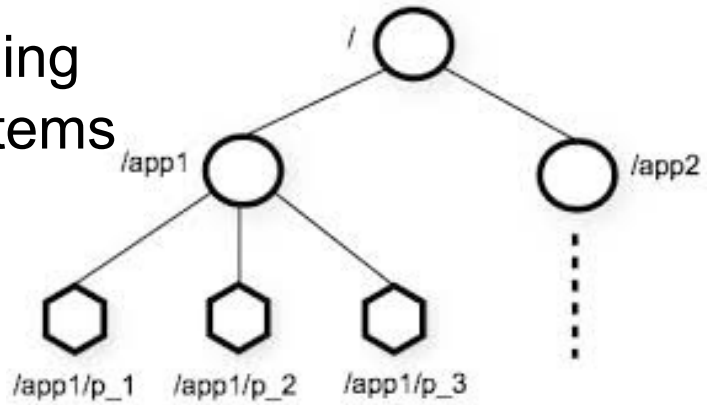


Additional ZooKeeper Use cases

- Configuration Management
  - Cluster member nodes Bootstrapping configuration from a central source
- Distributed Cluster Management
  - Node Join/Leave
  - Node Status in real time
- Naming Service – e.g. DNS
- Distributed Synchronization – locks, barriers
- Centralized and Highly reliable Registry

Summary of ZooKeeper

- An Open source, High Performance coordination service for distributed applications
- Centralized service for
  - Configuration Management
  - Locks and Synchronization for providing coordination between distributed systems
  - Naming service (Registry)
  - Group Membership
- Features
  - Hierarchical namespace
  - Provide Watcher on a znode
  - Allow to form a cluster of nodes
- Support a "large" volume of requests for data retrieval and update



Source : http://zookeeper.apache.org