ZooKeeper

A highly available, scalable, distributed, configuration, consensus, group membership, leader election, naming, and coordination service
Protocol Guarantees

1) Sequential Consistency - Updates from a client will be applied in the order that they were sent.

2) Atomicity - Updates either succeed or fail. No partial results.

3) Single System Image - A client will see the same view of the service regardless of the server that it connects to.

4) Reliability - Once an update has been applied, it will persist from that time forward until a client overwrites the update.

5) Timeliness - The clients view of the system is guaranteed to be up-to-date within a certain bound. Either system changes will be seen by a client within this bound, or the client will detect a service outage.
1) All servers store a copy of the data
2) A leader is elected at startup
3) Followers service clients, all updates go through leader
4) Update responses are sent when a majority of servers have persisted the change
All updates go through the leader where they are ordered and stamped with a monotonically increasing zxid.
1) UDP based
2) Server with the highest logged transaction gets nominated
3) Election doesn't have to be absolutely correct, just very likely correct
1) Each server initially nominate themselves
2) Servers poll each other to get their votes

* This is the currently implemented protocol Flavio has a better one in the works.
1) Each server initially nominate themselves
2) Servers poll each other to get their votes and vote for the one with the highest zxid if there isn't a winner
Leading

1) Leader does not lead until a quorum of followers have synced with it.

2) Zxid is a 64-bit number: 32-bit of epoch and 32-bit counter.

3) The first proposal from a leader is a NEWLEADER txn that has a zxid with the epoch bits one greater than the last logged zxid and the counter set to zero.

4) Leader accepts requests after a quorum have acked the NEWLEADER txn.

5) Everything processed in order.
ZooKeeper Servers

1) Forward Request
2) Send Proposal
3) Ack Proposal
4) Commit

Create a proposal and stamp with zxid

Log txn, but don't use until committed

Update in memory database and make visible
Anatomy of Standalone ZooKeeperServer

Data Tree

Request from Client

PrepRP  SyncRP  FinalRP

Session Manager

Respond to Client
Anatomy of Standalone ZooKeeperServer

1) Checks updates against the future
2) Creates a transaction

1) Logs txn
2) Releases request when synced to disk

1) Applies txn
2) Reply to client

Session Manager

Request from Client

Respond to Client
Anatomy of Leader ZooKeeperServer

Data Tree

Request from Client

PrepRP

SyncRP

Propose

Commit

Final

Followers

Session Manager

ACK tracker

Commit

Respond to Client

ACK

Propose to Client

Respond to Client

PrepRP

Commit

Final
Anatomy of Leader ZooKeeperServer

1) Sends a proposal to followers

1) Holds request until commit comes in
Anatomy of Follower ZooKeeperServer

- Request from Client
- Follow
- Commit
- Final
- Session Manager
- Respond to Client
- SyncRP
- Propose
- ACK
- Leader
- Data Tree
- Commit

SyncRP

Leader

Session Manager

Request from Client

Follow

Commit

Final

Respond to Client
Anatomy of Follower ZooKeeperServer

1) Forwards request to leader
1) DataNodes contain node data, stat, and child list
2) Hashtable maps path to DataNode
3) Updates logged to stable storage
4) Rough snapshots taken periodically
Snapshots

Sn = Snapshot at zxid n, Ln = Log started at zxid n
Current DataTree = Sn + Ln
Because we do not lock the Data Tree to snapshot, we get some txns in the snapshot that arrived after snapshot started. We have S'n = Sn + L'n, where L'n subset of Ln.
However S'n + Ln = Sn + L'n + Ln
Due to idempotent nature of the txns, L'n+Ln = Ln
Thus, S'n+Ln = Sn+Ln
1) A client must see the same view of the system no matter which server it connects to.

2) Followers are always consistent with leaders they may be behind in their updates.

3) Clients of a fast follower may have a more recent view than a slow follower.

4) Followers only serve clients if their view of the system is equal to or more up-to-date than the client's.

5) Client1 can connect to Server2, but Server2 will refuse Client2's connection until Server2 sees Zxid 16.
Protocol Guarantees

1) Sequential Consistency - Updates from a client will be applied in the order that they were sent.

2) Atomicity - Updates either succeed or fail. No partial results.

3) Single System Image - A client will see the same view of the service regardless of the server that it connects to.

4) Reliability - Once an update has been applied, it will persist from that time forward until a client overwrites the update.

5) Timeliness - The clients view of the system is guaranteed to be up-to-date within a certain bound. Either system changes will be seen by a client within this bound, or the client will detect a service outage.
Status

• Code in vault under yahoo/yresearch/projects/zookeeper
• Quorum and Standalone servers working
• Java and C clients available

Todo
• Convert server to use NIO
• More efficient follower syncing
• Check ACLs
• Perl, Python, and Ruby bindings
• Lots more testing!