Architecture of ZAB – ZooKeeper Atomic Broadcast protocol

[JUNE 20, 2015](https://distributedalgorithm.wordpress.com/2015/06/20/architecture-of-zab-zookeeper-atomic-broadcast-protocol/) [GUY MOSHKOWICH](https://distributedalgorithm.wordpress.com/author/guymoshkowich/)[7 COMMENTS](https://distributedalgorithm.wordpress.com/2015/06/20/architecture-of-zab-zookeeper-atomic-broadcast-protocol/#comments)

**Background**

ZooKeeper support clients reading and updating key values pairs with high availability. High availability is achieved by replicating the data to multiple nodes and let clients read from any node. Critical to the design of Zookeeper is the observation that each state change is incremental with respect to the previous state, so there is an implicit dependence on the order of the state changes. Zookeeper Atomic Broadcast (ZAB) is the protocol under the hood that drives ZooKeeper replication order guarantee. It also handles electing a leader and recovery of failing leaders and nodes. This post is about ZAB.

**Definitions**

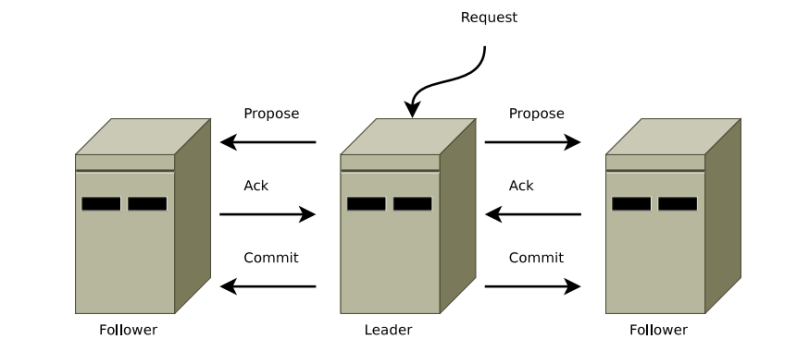
* leader and followers-  in ZooKeeper cluster, one of the nodes has a  leader role and the rest have followers roles. The leader is responsible for accepting all incoming state changes from the clients and replicate them to itself and to the followers. read requests are load balanced between all followers and leader.
* transactions –  client state changes that a leader propagates to its followers.
* ‘e’ – the epoch of a leader. epoch is an integer that is generated by a leader when he start to lead and should be larger than epoch’s of previous leaders.
* ‘c’ – a sequence number that is generated by the leader, starting at 0 and increasing. it is used together with an epoch to order the incoming clients state changes.
* ‘F.history’ – follower’s history queue. used for committing incoming transactions in the order they arrived.
* outstanding transactions – the set of transactions in the F.History that have sequence number smaller than current COMMIT sequence number.

**ZAB Requirements**

1. Replication guarantees
   1. *Reliable delivery –*If a transaction, M, is committed by one server, it will be eventually committed by all servers.
   2. *Total order –*If transaction A is committed before transaction B by one server, A will be committed before B by all servers. If A and B are committed messages, either A will be committed before B or B will be committed before A.
   3. *Causal order –*If a transaction B is sent after a 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.
2. Transactions are replicated as long as majority (quorum) of nodes are up.
3. When nodes fail and later restarted – it should catch up the transactions that were replicated during the time it was down.

**ZAB Implementation**

* clients read from any of the ZooKeeper nodes.
* clients write state changes to any of the ZooKeeper nodes and this state changes are forward to the leader node.
* ZooKeeper uses a variation of  [two-phase-commit protocol](https://distributedalgorithm.wordpress.com/2015/06/13/two-phase-commit/) for replicating transactions to followers. When a leader receive a change update from a client it generate a transaction with sequel number *c* and the leader’s epoch *e* (see definitions) and send the transaction to all followers. a follower adds the transaction to its history queue and send ACK to the leader. When a leader receives ACK’s from a quorum it send the the quorum COMMIT for that transaction. a follower that accept COMMIT will commit this transaction unless *c* is higher than any sequence number in its history queue. It will wait for receiving COMMIT’s for all its earlier transactions (outstanding transactions) before commiting.



picture taken from reference [4]

* Upon leader crashes, nodes execute a recovery protocol both to agree upon a common consistent state before resuming regular operation and to establish a new leader to broadcast state changes.
* To exercise the leader role, a node must have the support of a quorum of nodes. As nodes can crash and recover, there can be over time multiple leaders and in fact the same nodes may exercise the node role multiple times.
* node’s life cycle: each node executes one iteration of this protocol at a time, and at any time, a process may drop the current iteration and start a new one by proceeding to Phase 0.
  + ***Phase 0 –  prospective leader election***
  + ***Phase 1 – discovery***
  + ***Phase 2 – synchronization***
  + ***Phase 3 – broadcast***
* Phases 1 and 2 are important for bringing the ensemble to a mutually consistent state, specially when recovering from crashes.
* **Phase 1 – Discovery**  
  In this phase, followers communicate with their prospective leader, so that the leader gathers information about the most recent transactions that its followers accepted. The purpose of this phase is to discover the most updated sequence of accepted transactions among a quorum, and to establish a new epoch so that previous leaders cannot commit new proposals. Because quorum of the followers have all changes accepted by the previous leader- then it is promised that at least one of the followers in current quorum has in its history queue all the changes accepted by previous leader which means that the new leader will have them as well. [Phase 1exact algorithm available here.](https://distributedalgorithm.wordpress.com/?p=156&preview=true&preview_id=156)
* **Phase 2 – Synchronization**  
  The Synchronization phase concludes the recovery part of the protocol, synchronizing the replicas in the cluster using the leader’s updated history from the discovery phase. The leader communicates with the followers, proposing transactions from its history. Followers acknowledge the proposals if their own history is behind the leader’s history. When the leader sees acknowledgements from a quorum, it issues a commit message to them. At that point, the leader is said to be established, and not anymore prospective.[Phase 2exact algorithm available here.](https://distributedalgorithm.wordpress.com/?p=163&preview=true&preview_id=163&preview_nonce=ecb4461ca8)
* **Phase 3 – Broadcast**  
  If no crashes occur, peers stay in this phase indefinitely, performing broadcast of transactions as soon as a ZooKeeper client issues a write request. [Phase 3exact algorithm available here.](https://distributedalgorithm.wordpress.com/2015/06/20/algorithm-for-zab-phase-3-broadcast/?preview=true&preview_id=169&preview_nonce=318de41371)
* To detect failures, Zab employs periodic heartbeat messages between followers and their leaders. If a leader does not receive heartbeats from a quorum of followers within a given timeout, it abandons its leadership and shifts to state election and Phase 0. A follower also goes to Leader Election Phase if it does not receive heartbeats from its leader within a timeout.

**References:**

1. Flavio P. Junqueira, Benjamin C. Reed, and Marco Serafini, [*“Zab: High-performance broadcast for primary-backup systems”*](http://web.stanford.edu/class/cs347/reading/zab.pdf)
2. Andr´e Medeiros, [*“ZooKeeper’s atomic broadcast protocol: Theory and practice”*](http://www.tcs.hut.fi/Studies/T-79.5001/reports/2012-deSouzaMedeiros.pdf)
3. [*ZooKeeper Apache documentation*](http://zookeeper.apache.org/doc/r3.3.6/zookeeperInternals.html)
4. Benjamin Reed,Flavio P. Junqueira, [“*A simple totally ordered broadcast protocol”*](http://diyhpl.us/~bryan/papers2/distributed/distributed-systems/zab.totally-ordered-broadcast-protocol.2008.pdf)