HBase Book 0.89.0-SNAPSHOT

HBase Book 0.89.0-SNAPSHOT

Table of Contents

Chapter 1. Getting Started

Requirements

First...

Chapter 2. Data Model

Chapter 3. Implementation

Chapter 4. MapReduce

Chapter 5. Schema Design

Chapter 6. Shell

Chapter 7. Thrift

Chapter 8. REST

Chapter 9. Regions

This chapter is all about Regions.

Note

TODO: Review all of the below to ensure it matches what was committed -- St.Ack 20100901

Region Transitions

Regions only transition in a limited set of circumstances.

Cluster Startup

During cluster startup, the Master will know that it is a cluster startup and do a bulk assignment.

Note

This should take HDFS block locations into account.

- Master startup determines whether this is startup or failover by counting the number of RegionServer nodes in ZooKeeper.
- Master waits for the minimum number of RegionServers to be available to be assigned regions.
- Master clears out anything in the `/unassigned` directory in ZooKeeper.
- Master randomly assigns out `-ROOT-` and then `.META.`.
- Master determines a bulk assignment plan via the `LoadBalancer`
- Master stores the plan in the `AssignmentManager`.
- Master creates `OFFLINE` ZooKeeper nodes in `/unassigned` for every region.
- Master sends RPCs to each RegionServer, telling them to `OPEN` their regions.

All special cluster startup logic ends here.

Note

So what can go wrong?

- We assume that the Master will not fail until after the `OFFLINE` nodes have been created in ZK. RegionServers can fail at any time.
- If an RS fails at some point during this process, normal region open/opening/opened handling will take care of it.

  If the RS successfully opened a region, then it will be taken care of in the normal RS failure handling.

  If the RS did not successfully open a region, the RegionManager or MasterPlanner will notice that the OFFLINE (or OPENING) node in ZK has not been updated. This will trigger a re-

assignment to a different server. This logic is not special to startup, all assignments will eventually time out if the destination server never proceeds.

- If the Master fails (after creating the ZK nodes), the failed-over Master will see all of the regions in transition. It will handle it in the same way any failed-over Master will handle existing regions in transition.

Load Balancing

Periodically, and when there are not any regions in transition, a load balancer will run and move regions around to balance cluster load.

- Periodic timer expires initializing a load balance (Load Balancer is an instance of `Chore`).
- Currently if regions in transition, load balancer goes back to sleep.

  ### Note

  Should it block until there are no regions in transition.

- The `AssignmentManager` determines a balancing plan via the LoadBalancer.
- Master stores the plan in the `AssignmentMaster` store of `RegionPlan`s
- Master sends RPCs to the source RSs, telling them to `CLOSE` the regions.

That is it for the initial part of the load balance. Further steps will be executed following event-triggers from ZK or timeouts if closes run too long. It's not clear what to do in the case of a long-running CLOSE besides ask again.

- RS receives CLOSE RPC, changes to CLOSING, and begins closing the region.
- Master sees that region is now CLOSING but does nothing.
- RS closes region and changes ZK node to CLOSED.
- Master sees that region is now CLOSED.
- Master looks at the plan for the specified region to figure out the desired destination server.
- Master sends an RPC to the destination RS telling it to OPEN the region.
- RS receives OPEN RPC, changes to OPENING, and begins opening the region.
- Master sees that region is now OPENING but does nothing.
- RS opens region and changes ZK node to OPENED. Edits .META. updating the regions location.
- Master sees that region is now OPENED.
- Master removes the region from all in-memory structures.
- Master deletes the OPENED node from ZK.

The Master or RSs can fail during this process. There is nothing special about handling regions in transition due to load balancing so consult the descriptions below for how this is handled.

Table Enable/Disable

Users can enable and disable tables manually. This is done to make config changes to tables, drop tables, etc...

Note

Because all failover logic is designed to ensure assignment of all regions in transition, these operations will not properly ride over Master or RegionServer failures. Since these are client-triggered operations, this should be okay for the initial master design. Moving forward, a special node could be put in ZK to denote that a enable/disable has been requested. Another option is to persist region movement plans into ZK instead of just in-memory. In that case, an empty destination would signal that the region should not be reopened after being closed.

Disable

- Client sends Master an RPC to disable a table.
- Master finds all regions of the table.
- Master stores the plan (do not re-open the regions once closed).
- Master sends RPCs to RSs to close all the regions of the table.
- RS receives CLOSE RPC, creates ZK node in CLOSING state, and begins closing the region.
- Master sees that region is now CLOSING but does nothing.
- RS closes region and changes ZK node to CLOSED.
- Master sees that region is now CLOSED.
- Master looks at the plan for the specified region and sees that it should not reopen.
- Master deletes the unassigned znode. It is no longer responsible for ensuring assignment/availability of this region.

Enable

- Client sends Master an RPC to disable a table.
- Master finds all regions of the table.
- Master creates an unassigned node in an OFFLINE state for each region.
- Master sends RPCs to RSs to open all the regions of the table.
- RS receives OPEN RPC, transitions ZK node to OPENING state, and begins opening the region.
- Master sees that region is now OPENING but does nothing.
- RS opens region and changes ZK node to OPENED.
- Master sees that region is now OPENED.
- Master deletes the unassigned znode.

RegionServer Failure

- Master is alerted via ZK that an RS ephemeral node is gone.
- Master begins RS failure process.
- Master determines which regions need to be handled.
- Master in-memory state shows all regions currently assigned to the dead RS.
- Master in-memory plans show any regions that were in transitioning to the dead RS.
- With list of regions, Master now forces assignment of all regions to other RSs.
- Master creates or force updates all existing ZK unassigned nodes to be OFFLINE.
- Master sends RPCs to RSs to open all the regions.
- Normal operations from here on.

There are some complexities here. For regions in transition that were somehow involved with the dead RS, these could be in any of the 5 states in ZK.

- `OFFLINE` Generate a new assignment and send an OPEN RPC.
- `CLOSING` If the failed RS is the source, we overwrite the state to OFFLINE, generate a new assignment, and send an OPEN RPC. If the failed RS is the destination, we overwrite the state to OFFLINE and send an OPEN RPC to the original destination. If for some reason we don't have an existing plan (concurrent Master failure), generate a new assignment and send an OPEN RPC.
- `CLOSED` If the failed RS is the source, we can safely ignore this. The normal ZK event handling should deal with this. If the failed RS is the destination, we generate a new assignment and send an OPEN RPC.
- OPENING or OPENED If the failed RS was the original source, ignore. If the failed RS is the destination, we overwrite the state to OFFLINE, generate a new assignment, and send an OPEN RPC.

In all of these cases, it is important to note that the transitions on the RS side ensure only a single RS ever successfully completes a transition. This is done by reading the current state, verifying it is expected, and then issuing the update with the version number of the read value. If multiple RSs are attempting this operation, exactly one can succeed.

Master Failover

- Master initializes and finds out that he is a failed-over Master.
- Before Master starts up the normal handlers for region transitions he grabs all nodes in /unassigned.
- If no regions are in transition, failover is done and he continues.
- If regions are in transition, each will be handled according to the current region state in ZK.
- Before processing the regions in transition, the normal handlers start to ensure we don't miss any transitions. The handling of opens on the RS side ensures we don't dupe assign even if things have changed before we finish acting on them.
    - OFFLINE Generate a new assignment and send an OPEN RPC.

- CLOSING Nothing to be done. Normal handlers take care of timeouts.
- CLOSED Generate a new assignment and send an OPEN RPC.
- OPENING Nothing to be done. Normal handlers take care of timeouts.
- OPENED Delete the node from ZK. Region was successfully opened but the previous Master did not acknowledge it.

- Once this is done, everything further is dealt with as normal by the RegionManager.

Summary of Region Transition States

Note

Check below is complete -- St.Ack 20100901

Master

- Master creates an unassigned node as OFFLINE.

  Cluster startup and table enabling.

- Master forces an existing unassigned node to OFFLINE.

  RegionServer failure.

  Allows transitions from all states to OFFLINE.

- Master deletes an unassigned node that was in a OPENED state.

  Normal region transitions. Besides cluster startup, no other deletions of unassigned nodes is allowed.

- Master deletes all unassigned nodes regardless of state.

  Cluster startup before any assignment happens.

RegionServer

- RegionServer creates an unassigned node as CLOSING.

  All region closes will do this in response to a CLOSE RPC from Master.

  A node can never be transitioned to CLOSING, only created.

- RegionServer transitions an unassigned node from CLOSING to CLOSED.

  Normal region closes. CAS operation.

- RegionServer transitions an unassigned node from OFFLINE to OPENING.

  All region opens will do this in response to an OPEN RPC from the Master.

  Normal region opens. CAS operation.

- RegionServer transitions an unassigned node from OPENING to OPENED.

Normal region opens. CAS operation.

Appendix A.