Region management in Master

This document describes managing master-controlled resources, in a manner that is reliable, scalable, and sane. It purports to start with regions, however, the same scheme may also be applicable to tables. Currently, other approaches are already used for tables, so that may not be necessary.

Table of Contents

Problem

- Current region management code is hard to reason about.
  - We historically find lots of bugs in code that manages regions (assignment manager, shutdown handling, etc.).
  - For the cases like split, merge, alter table, etc., we have separate logic to ensure consistency; these logics overlap, e.g. we have to see if regions are merging when we want to reassign them, handle parallel reassignment and split, etc.
- The steps were recently made to have better management of the tables via table locks; however

- it's not clear if locks will scale, if reused for regions; especially in sufficiently large clusters where region opens/moves/splits can be frequent;
- table locks themselves may also block table updates for unacceptable amount of time, especially in case of hardware/power/etc. failures where lock will not be released until session is closed in ZK after connection times out. This problem also increases with scale.

Design principles

The proposed approach to solve the above issues is as follows.

Table record

For every table, we maintain a single persistent atomically modifiable record. As general table management is out of the scope of this document, this record only stores one thing – table "version" that is bumped on every table-wide operation like alter or disable. This allows us to apply table alterations "lazily" to regions currently in transition/split/merge/offline.

Note that the logic around version is applicable for the current "reopen regions on change" approach; if regions could be updated w/o reopen it could be done in a more simple manner.

Region record

For every region, we maintain single persistent atomically modifiable record. This record is the only source of truth about the region, and is never removed while the region is relevant. This simplifies current situation where ZK state, master state and META state can all conflict in various special ways; as well as the special cases with splitting/merging/etc. that are caused by having separate transient nodes.

Other entries may exist (e.g. , if we use ZK, META can exist for clients because ZK is inconvenient to query) but they are never used as truth during normal operations, only as cache/backup mechanism/etc.

The record has a few things. First, it has region state as described by region state machine (see below). Everyone (master, RS) uses the same state machine - local memory state follows the state from the above source via change notifications, or after successfully performed record changes.

Then, the record contains table version with which this region was last opened, or should be opened if not currently opened. The version doesn't allow any sort of "time travel" to previous versions – region internals will be initialized without regard for table version. It only allows us to detect when region is out of date.

Finally, the record may contain some additional information (for example, server name when Opened or Opening, last error if Closed, or daughter IDs when Splitting, etc.).

Operation semantics/messages

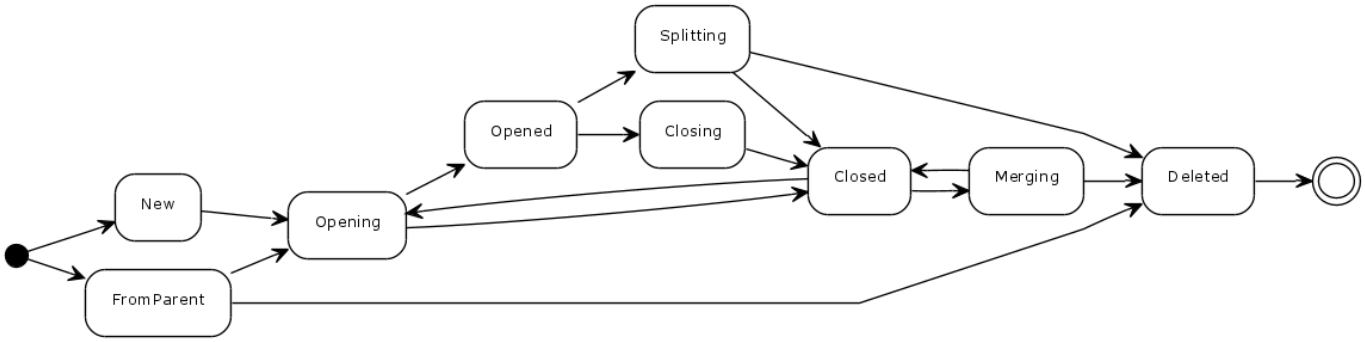
The state machine above also supersedes master/RS messages; for example, if master asks RS to open a region that RS is already opening, during recovery, it should not cause any error; message to RS is treated as "optimization" over state machine state that should be followed by RS, and sending multiple is not an error.

Hypothetically, given that all the operations depend on this record anyway (like they do now on transient ZK nodes), and given how we already use ZK notifications to detect transitions, we could make the record the main communication mechanism – the only message master would send to server would be idempotent "watch node for this region", as an optimization to not have all RSes watch all regions. That could simplify things, at the cost of increasing the number of ZK notifications. Not clear if that will become a bottleneck.

Region state machine

Proposed region state machine is approximately as such. Notes:

- Some states have been collapsed (e.g. FailedToOpen and Closed/Offline are the same things, except for additional information like error code).
- FromParent state is used both for merge and split for newly created daughters (see below).
- Deleted state is used when RS deletes the region, to notify master of the necessary cleanup.



Example logic around the state

Open region

1. Master updates state to Opening, and sends RS a message.
2. RS remembers the table version from the state, and starts opening the region.
3. RS opens the region.
    a. If state did not change, it updates current state to Opened.
    b. If table version changed, repeats the open starting at (2).

    c. If the node is inconsistent (master hijacked the region), abandons/cleans up the open.

Close region

1. Master updates the region to Closing and sends message to RS.
    a. For recovery and in case of send failure, master can get all transition nodes and send messages again.
2. RS closes the region and updates the state to Closed.

SSH region

Close and open region, with close performed directly by master.

Split region

1. RS creates new records for regions in FromParent state, and updates the parent to Splitting (the records are necessary for concurrent alters to work, see "alter table").
2. RS closes the parent and creates the daughters.
    a. If one of these operations fails, RS changes parent to Closed, changes daughters' state to Deleted, and reopens the parent.
    b. If RS itself fails during the operations, master changes parent state to Closed, deletes daughter records, and puts parent thru normal assignment.
3. RS updates parent state to Deleted, and daughter record state to Opening. This commits the split.
4. After that daughters are open, and behave, as normal regions.

Merge regions

1. Master closes both regions as described above.
2. Master sets both regions' state to Merging and creates the new region record in FromParent state (the record is necessary for concurrent alters to work, see "alter table").
3. Master merges region contents in HDFS.
    a. If that fails, new region state is set to Deleted, and old regions are both set back to Closed and are re-opened normally.
4. Master sets both regions' states to Deleted, and new region state to Opening. This commits the merge.
5. Master opens the new region on requisite RS normally.

Alter table

In current implementation, to alter table, all regions have to be reopened.

1. After all the updates like table descriptor are done, master bumps the table version.
    a. If master fails here, on recovery discrepancy between table and regions' versions can be detected and alter.

2. Master goes thru the list of regions.
    a. For opened regions, the region is set to Closing w/version bump, then reopened as described above.
        i. This could also be simplified by bumping the version only and sending message to RS, RS can cycle the region internally, closing it and then opening.
    b. Regions that are not Opened just have their version bumped.
        i. For example, for concurrent reassignment, when server finishes opening region and sees version discrepancy, it aborts the open and does another one.

Note that with some additional checks in (2), several alter-s could be enabled to proceed in parallel, provided that overwriting table descriptor/etc. is done correctly.

Implementation details

It is suggested that ZK is used for records. It both has convenient model (asynchronous APIs, notifications, multi-node atomic updates), and is more scalable and fault tolerant than a single META server that is also serving other regions.

META would still exist, and would be a mirror of ZK state for two purposes:

- Clients finding regions (ZK doesn't allow keyBefore query that is used for META).
- Recovery in case of cluster move or some other situation where ZK is completely wiped.