HBASE Region Assignment Manager Improvement (V4)

Draft V0.0.2

1. Introduction

Assignment Manager (AM) is a component in the Apache HBASE Master that manages regions to Region Servers (RS) assignment. It ensures that any one region is assigned to just one RS.

This is the fourth attempt to make assignment manager more reliable. The first attempt (AM v1) was an in-memory only implementation pre-0.90 release. In 0.90 release, the second implementation (currently used in 0.98.x and 1.x releases) of AM (AM v2) uses Zookeeper (ZK) – RS uses ZK to notify the Master about region assignment progress. ZK is also used as a store for transient region state so that when the Master dies, a new Master could pick up where the old Master left off. Once a region is open, the RS updates the META system table with the new region location. We have 3 places (!) to store the region states: Master in-memory cache, Zookeeper, and the META table. To keep all 3 places in-sync is pretty complicated (or mission impossible). We saw a lot of bugs coming from that (see Jimmy Xiang's Apache HBASE AssignmentManager Improvements blog in Nov 2012).

In the Master branch (for 2.0 release), we have the third implementation of AM (AM v3), called *ZK-less Region Assignment* (AKA assigning regions without involving ZooKeeper - HBASE-11059). This approach adopted some discussion from HBASE-5487 (Generic framework for Master-coordinated tasks). It simplifies the logic, achieves greater scale, and improves the speed of assignments. Besides getting rid of ZK, the most significant improvement is that MASTER is the only actor to write to the META table in all assignment situations. To guarantee the state consistency and for atomic state update, the requirement is to have the META table co-located on Master. Again, Jimmy Xiang has a great blog talking about this implementation in details.

In this document, we propose an approach to further enhance the Assignment Manager (AM v4). The goal is to make AM more stable and has less issue to worry. The complete (re-) implementation of cluster management is not in scope of this document; however, when we improve the AM (which focuses on RIT part of cluster management), we need to keep in mind of its overall impact on cluster management (especially state changes in various components of Master - table states, a list of regions, cluster states such as dead RS lists).

2. Problems:

(1). While Co-locate META table with Master give atomic update on region state in AM v3, it has some drawbacks:
(a). It complicates the AM logic during MASTER failover. Once Master failover happens, we have to move META or have some special assignment logic;
(b). Some future improvement like split META is difficult to accomplish with co-locate. With co-locate, if the META table splits, we need to have special logic to force all regions hosted by the Master server.
(c). Always-colocate is no-no for some customers.
(d). Meta table might end up hosting way more data (file layout/namespace) than it is doing today, making it impossible to be hosted by a single region server.

(2). Even with the simplification of removing ZK, we still have too many states to track and the complexity of code caused potential bugs.

(3). Hard to scale to 1 million regions and beyond (see https://docs.google.com/document/d/1eCuqf7i2dkWHL0PxcE1HE1nLRQ_tCyXI4JsOB6TAk60/edit#heading=h.80vcerzbkj93 for detailed discussion). With too many state changes in META and no META split allowed, it is hard to perform good when we have a lot of regions to manage (assign/unassign/move).

3. Scenarios

Region assignment operations have the following external triggers:

(1). Master Initialization.

(2). Shell commands

(2.1). Table commands: create, disable, enable, truncate.

(2.2). Region commands: assign, close, unassign, move, merge, split

(3). Balancer (regular or from shell).

(4). RS recovery ("server shutdown handler (SSH)").

(5). Periodic check (for RS operations that take too long, e.g. opening/closing).

4. Design and code considerations

The design should:
(1). Consistent: Have one source of truth and avoid split-brain scenario

(2). Recoverable: Be resilient to master or RS failures
(3). Not lose functionality compared to current AM (eg. Bulk region assignment)
(4). Performant and scalable (eg. handle 1M region assignment fast)
(5). (?) Rolling Upgradable from 1.x to 2.0 release.

The code should
(1). Correctness: not cause corruption/inconsistency
Enis pointed out several scenarios that we need to be really careful (bugs in the past or have not handled well)
(a). RS crashed/stopped/shutted down, before SSH; if the Master sends RS region close request and got exception, Master should wait for SSH to run (not assuming region is closed cleanly)
(b). Concurrent region move and split
(c). Concurrent region assign/move/close and table DDL
(d). Region move/assign while RS crash
(2). Understandable: easy to read from human being
(3). Debuggable: when problem happens, easy to debug and troubleshooting
(4). Testable and verifiable: easy to unit test
(5). Isolatable: the logic should be isolated from the rest of the Master cluster managment.

5. Implementation choices and details

HBASE-5487 (Generic framework for Master-coordinated tasks) has a detailed investigation on different approaches. For example, for the persistent storage, it considered co-locate system table, non-co-located system tables, ZK, and customer master WAL. The JIRA also did investigation on applying Fault Tolerant Executor (FATE) model from the Apache Accumulo project.

Since the discussion on HBASE-5487 2 years ago, we implemented HBASE-12439 (Procedure V2, in short Proc-V2) in HBASE-1.1 release. The approach is similar to Accumolo FATE. It has a procedure store to store intermediate states for later steps/stages (eg. support multiple stages – such as parent waiting for child procedure to complete before continuing) and for crash recovery (eg. MASTER failover). The first phase of Procedure V2 moved table DDLs (Create/Alter/Truncate/Delete/Enable/Disable table and Add/Modify/Remove column family) from handlers to procedure-based. This improves reliability and avoids corruption due to failure in the middle of DDL operation. The work also lays a ground for converting other multiple-steps/stages Master operations (eg. snapshot) in the future.

The Proc-V2 design is very suitable to solve the AM co-locate problem. In this document, we will focus on AM improvement based on Proc-V2.

Most assignment work done by the Master is driven by 3 events: (1). RS online events; (2). RS offline events; and (3). Assignment events.

The Proc-V2 executor has a “scheduling queue” that will use the events from ServerManager or AM to trigger the procedure execution.

All assignment-related events could be broken down to (1). online a region (assign/open or reopen), (2). close a region (unassign; the region would be online in a same or likely different RS later), (3). Offline a region (another type of unassign, but the region will not be re-opened automatically unless triggered by user online event), (4). merge two regions (create a new region in a RS for merging 2 regions to one), and (5). Split one region to two region (split one region in a RS to 2 smaller regions). For ‘move a region to different RS’, it is basically calling close the region in RS1 and open the region in RS2. Same is true for balancer.

(1). Assignment States

We leverage Proc-V2 to simplify region assignment. It is natural that Proc-V2 would take care ‘State machine’ part of current AM.

In Proc-V2, we will get rid of excessive state changes in META and rely on procedure store (WALStore - WAL is the current Proc-V2 procedure store implementation) to deal with intermediate state of regions. Because of procedure store, we have ‘fence’ to avoid “double assignment” issue that we see currently in 1.x release.

There is no need to have intermediate states (such as PENDING_OPEN) in the META table. In META table, only final states (OPEN, CLOSE, OFFLINE) are stored. We always can recover intermediate states of a region from WALStore.

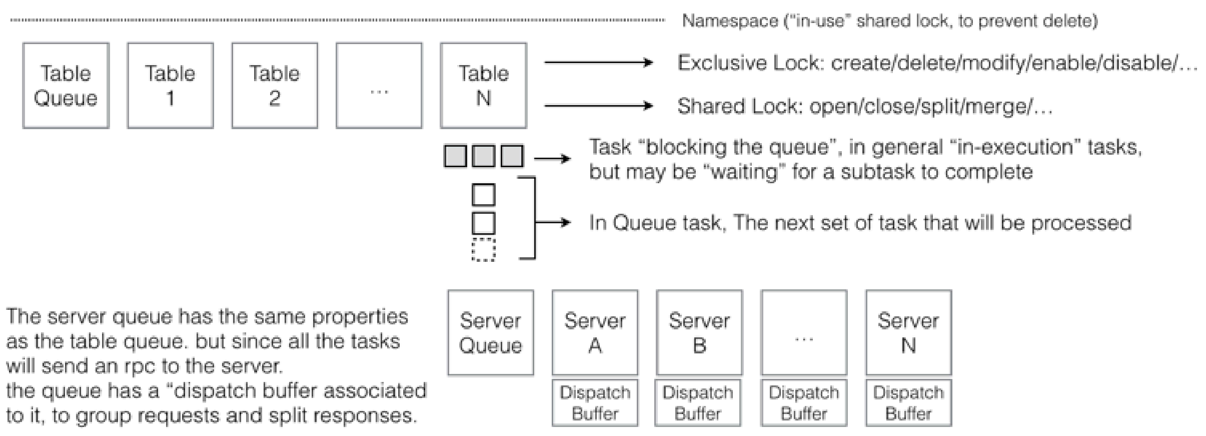
TODO: we need to determine when to recover procedure during MASTER failover or during MASTER restarts.

Summary: state machine logic move to Proc-V2; intermediate states stored in procedure store.

(2). Procedure Queues (Table and Server Queues)

In Phase-1 implementation of Proc-V2, we have table queues to deal with table DDLs for each table. In AM v2.5, we will use both table and server queues – each table queue deals with a particular table, and each server queue works with a particular RS.

The table queue uses table name to identify the queue; the server queue uses [server name, port, timestamp] to uniquely identify one RS. If a RS re-starts, the timestamp would change. We will delete the old server queue and create a new server queue with the new timestamp.


Namespace ("in-use" shared lock, to prevent delete)
Table Queue
Table 1
Table 2
...
Table N
Exclusive Lock: create/delete/modify/enable/disable/...
Shared Lock: open/close/split/merge/...
Task "blocking the queue", in general "in-execution" tasks, but may be "waiting" for a subtask to complete
In Queue task, The next set of task that will be processed
The server queue has the same properties as the table queue. but since all the tasks will send an rpc to the server. the queue has a "dispatch buffer associated to it, to group requests and split responses.
Server Queue
Server A
Server B
...
Server N
Dispatch Buffer

The high-level flow is:
- All (un)assignment (open/close/merge etc.) operation always starts from a Table queue (a table DDL procedure)
- Inside the table DDL procedure, it would start child assignment procedures (server procedures). Depending on how many RS will be involved in the assignment; multiple server queues could be involved.
- Each server procedure would be put into a Server Queue.
- When a Server Queue picks a procedure to execute, it would issue a RPC to RS.
- Internally, the RPC goes to a Dispatcher to batch multiple mutually exclusive procedures for the same server to one RPC call.
- The RS receives the batch RPC and then executes them parallel. The RS does not wait for the entire RPC finish to send response. When some part of request completes, the RS would send response to Master.
- The Master would receive the response from RS, then continue the Server procedure.
- Once the Server procedure is done, it would return to Table procedure with the result (assigned, failed, or exception throw).
- Based on results of Server procedures (child procedures), the table procedure determines whether it would complete, or re-assign one or more region to a different RS (see SSH section below for some consideration).

Note: the assignment plan logic should be outside procedure code. We should leverage the existing assignment plan logic (no change here). The table procedure code just calls assignment plan code to find out the assignment details and then use the plan to determine which Service queue to use.

At the beginning, the table procedure is in RUNNABLE state. Once the table procedure adds server procedures (child procedures) and child procedures start to execute, the parent table procedures changes to WAITING state. It would go back to RUNNABLE states once all child procedures completes (either successful or rollback). (*Note: the logic should already existed in the 1.x release, but we don't have a use case before, need to test it and check whether anything is missing.)*

[TODO: future consideration: one server queue per WAL for scale improvement.]

Summary: using table/server queues as scheduling mechanism: server queues for region assignment on a RS; all assignment initiated from table queues.

(3). Region commands

We need to convert existing assign/unassign/move/merge/split requests to Proc-V2 based. Then all procedures would come down to going through table/server queues.

The table DDLs (eg. Create/Alter/Truncate/Delete/Enable/Disable table and Add/Modify/Remove column family) should use the new assignment procedures (instead of calling AM). The assignment procedures are child procedures of table DDLs.

The same is for balancer (either from shell or from ServerManager). It is a little complicated, as it deals with multiple tables in multiple server queues. At the end, it just consists of multiple table assignment DDLs.

Summary: All shell commands became a procedure and internally share the same code with the system initiated assignment operations.

(4). Locking

In 1.x release, we have exclusive table lock on table DDL procedures. The exclusive locks (X-lock) store in Master memory and in ZK. The ZK storage of lock is for backwards compatible and also for operations like assignment to take shared table lock (S-lock). Given all operations moving to Proc-V2, there is no need to have locks stored in ZK.

For table DDL, we will have X table lock; for region assignment, we will have S table lock – this would prevent table DDL while regions are in transition.

In addition, we will have region lock (no need to have X or S lock distinction, as AM is the only consumer here) – this is to prevent concurrent region operation on the same region (eg. assign a region, while at the same time split this region).

The lock request comes from the table DDL operations. The table queue has acquireExclusiveLock() and acquireSharedLock([resources]) that can be called by the procedure. We don't have to do any locking in Server procedure queue.

Here is the high-level pseudo code:

```
public synchronized boolean trySharedLock, pass in a list of regions to lock  (eg. merge is 2
regions; assign/unassgn is 1 region)
        for each region R
                if locked return false
        for each region R
                lock R
        share lock T
```

We will store table and region locks in memory. Given we could have millions of locks, a hash table would be used to fast look up [table, region] lock.

[TODO: where namespace lock fit? We don't have NS lock today]

Summary: Introducing region-level lock in TableRunQueue

(5). Assignment Procedures (server procedures)

The assignment procedure has 5 types: (1) Open, (2) Close, (3). Offline, (4) Merge, (5) Split.

Offline and close should be no different from RS point of view, only Master needs to know the difference and update the META table/in-memory state accordingly.

[TODO: should we treat split just open two new regions and offline the parent region? Same as Merge – open one new region and offline 2 parent region – should RS knows whether the

operation is for merge/split or just master knows the ultimate goal and RS does not need to know details of operation]

The AssignmentProcedure(RegionInfo) from the Master side will
- SendAssignmentRPC – inside implementation, it actually sends the request through a dispatcher (a buffer, see the Dispatcher section below), not directly to RS.
- WaitForResponse – the procedure need to set its state to WAITING_TIMEOUT state (the default timeout value is 10 seconds).

[TODO: we need to determine the optimistic timeout value.]
[TODO: we need to determine how many retries to do before failing the procedure. Note: if RS is dead, ServerManager might detect it and force rollback of the procedure – see SSH section]
- If region is successfully open/close/offline from RS, update the META table and in-memory state in the MASTER for the new state and return succeed; otherwise, return failure.

On the RS side, it just executes the RPC as usual; the only difference is that RS don't need to report intermediate states back to master. Also there is no need to store intermediate state in RS; if RS dies, Master would retry in a different RS (See SSH section below for some consideration). If Master dies, the new Master would retry using the same procId and RS could just ignore it.

For bulk assignment (batching from dispatcher), the RS does not need to finish the entire RPC request and send response back. It could send partial response back.

Summary: implementing server-based assignment procedure to deal with open / close / offline / merge /split region operation.

(6). Dispatcher

The dispatcher is used for bulk assignment functionality. For performance reason, we want to batch as many mutually exclusive RS requests as possible to 1 RPC. Mutually exclusive means that procedures do not touch the same region (eg. assign R1 and unassign R2 are maturely exclusive procedures; merge R1, R2 and unassign R1 are NOT mutually exclusively and has to be execute in-order). The region lock in table procedure level would deal with the mutual exclusive part; once the procedures in server queues, all server procedures in the same server queue would be mutually exclusive and can be batched.

The dispatcher will try to batch all assignment procedures against the same RS to 1 RPC. It would send AssignmentBatchRequest to RS.

[TODO: investigate whether Protocol Buffer support List<ActionObject>, where ActionObject could be OpenAction, CloseAction, MergeAction – different action has different attributes; two alternatives: ActionObject contains all possible attributes, depending on action type, some attributes could be empty/optional; or we have multiple lists in the protoBuf – List<OpenAction>, List<CloseAction>, List<MergeAction> etc. - some lists might be empty]

The RS receives the AssignmentBatchRequest. It would parse it and execute each action inside the request parallel. Once some action completes (not waiting for the entire batch completes), it would notify Master about the result. Both request and response contains ProcId, so RS knows whether all actions from a particular procedure is completed; and the receiver from the Master side would know whether it is good enough to send the response to the assignment procedure (server procedure).

[TODO: investigate more on this: the receiver from the Master side could be part of dispatcher or just part of HMaster.]

Summary: in the ServerRunQueue, add dispatcher to support batch several assignment procedure for the same RS to 1 RPC.

(7). Server Shutdown Handler (SSH)

When Master send a assignment request (either assign/open or unassign/close) to RS, if RS does not respond (either RS down or network partition), Master cannot assume that region is not touched and try to find a different RS to assign. Instead, Master needs to

- wait and retry
- force expire the RS and let SSH run and complete

Only after SSH completes, the Master can find a different RS to assign.

The reason is that without SSH, the presumed dead RS might still alive and do something on the region (eg. cleaning up region /.tmp directory – when the real RS who hosts the region tries to do some operation and add files in .tmp directory, it would find that the ./tmp directory suddenly disappeared).

Note: even SSH is run, the only thing that we can guarantee is that the presumed dead RS could not write to WAL. It really has **no** guarantee that client talk to dead RS for stale data or the dead RS clean up /.tmp directory if it does not know that itself is killed by the Master.

As part of the SSH, it would delete the server queue for the dead server. The server queue deletion would force all assignment procedures under the server queue to abort.

At the end of SSH, it would try to re-open the regions hosted by the dead RS to other RS. (Note: even the dead RS restarts, due to different timestamp, we treat it as a new RS.) the reassign/reopen operation is similar to balancer (regions from multiple tables re-assign to multiple servers). *[TODO: more details on the design. It could be shared by the balancer operation.]*

[TODO: add more details on updated SSH to support new AM]

(8). Master Initialization

When the MASTER restarts, we need to make sure that regions are assigned based on the orders in procedure store.

Not all tables are equivalent. The META table is the most important table, we need to assign it first; then system tables, then user tables. Even for non-META system tables, some are more important than others. For example, ACL system table is more critical to the cluster than the BACKUP system table (part of BACKUP effort currently in execution). *[TODO: is it worth the effort to create multiple classes of system tables? Or just treat all non-META system table the same to simplify the design.]*

[TODO: investigate and add more details.]

(9). HBCK

[TODO: What to do with RIT – HBCK should know whether the assign procedure is running.]

(10). Rolling upgrade

The rolling upgrade support should be possible – we upgrade RS one-by-one (RS should understand both existing assignment RPC and new assignment request). Once all RS are upgraded to 2.0. We then upgrade MASTER. Once MASTER restarts, it should check META table and resend assignment request to RS using Proc-V2.

[TODO: more details on step-by-step upgrade.]

6. Work Items breakdown

(1). Server Queue (when St.ack re-wrote SSH based on Proc-V2 in HBASE-13616, he already had some implementation. Take a look at hbase-server/src/main/java/org/apache/hadoop/hbase/master/procedure/ServerCrashProcedure.java)
(2). (Table, Region) Lock (we have Table lock and need to expand to lock region in memory; given everything going though MASTER, table lock in ZK might not be needed.)
(3). Procedure Queue improvement (eg. Procedure execute - once child is added and running, parent goes to WAITING state.)
(4). Implement AssignProcedure from Master-side
(5). Implement AssignProcedure from RS-side (we can reuse most of RS side AM logic)
(6). A Dispatch-implementation to batch operations (batch multiple assignment and unassignment on one RS to 1 RPC)
(7). Inside table DDL, if any DDL involves region assignment, instead of calling AM, it should call Assignment procedures.
(8). Update/rewrite SSH to support new AM
(9). Implement move/split/merge shell commands to be Procedure-based
(10). Implement balancer to be Procedure-based
(11). Incorporate the Proc-V2 code so that the Master would do assignment based on new Proc-V2
(12). HBCK
(13). Rolling upgrade
(14). Clean up existing AM code