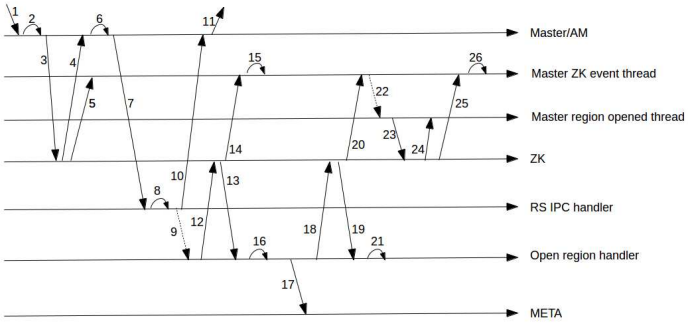
ZK-less Region Assignment (HBASE-11059)

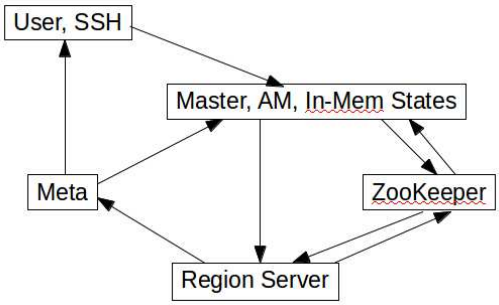
The Problem

Currently, region assignment uses ZK to do communication between master and region servers. Many steps are involved in one region assignment:



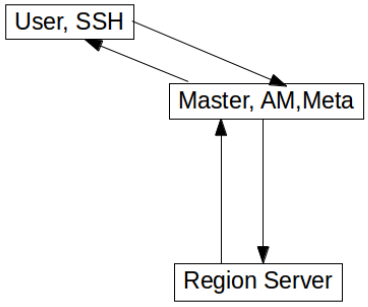
It is error-prone and hard to maintain, scale.

We have region assignment information in ZK, master memory, and meta table. They are not consistent during region transitions. Both master and region servers can update meta and ZK.



Propose

With HBASE-10569, we co-located meta and master. To further improve it, we have HBASE-11059 (ZK-less region assignment). The goal is to simplify the region assignment a lot to improve the performance and make it more scalable.



In this approach,

1. Region states are stored in meta, and cached in master memory, and they are consistent all the time.
2. Only master update the meta table. Region server doesn't update it.
3. Master and region server talk to each other directly on region assignment.

The call flow this approach is much simpler.

1. Region open/close should be initiated by master
    a. Master sets the region to pending_open/pending_close after sending the corresponding request to the region server

Comment [1]: ... because only the master is editing the meta (I see you say this next -- sorry)? Because the meta is on the master so we can do atomic meta/memory updates?

Comment [2]: Since they are colocated, we'd like the update to be atomic

Comment [3]: Forgive me if I'm asking something obvious - but would this in-memory update of meta still go thru RPC or it will just go directly?

Comment [4]: We assume the meta is colocated with the master. If so, we can make a short-circuit call instead of going through RPC.

Comment [5]: Yep, I think so. I'm just keeping in mind multi-master model, but since it's separate efforts, I'm thinking of what hookup points it'd be good to have, so it's easier to "intercept" such update for consensus also.

Btw, meta is now always colocated with master, it's not an option user may choose to turn off?

Comment [6]: In the meta table? In a new column on the meta?

Comment [7]: Yes

b. Region server reports back to the master after open/close is done (either success/failure, for close, in case failure, the region server can just abort as it does today)
c. If region server has problem to report the status to master, it must be because the master is down or temporary network issue. Otherwise, the region server should abort since it must be a bug. If the master is not accessible, the region server should keep trying until the server is stopped or till the status is reported to the (new) master
d. If region server dies in the middle of opening/closing a region, SSH picks it up and finishes it somewhere else
e. If master dies in the middle, the new master recovers the state during initialization from the meta table. It should get the report from the region server soon.

2. Region split/merge should be initiated by region servers

a. To split a region, a region server sends a request to master to try to set a region to splitting, together with two daughters to splitting new. If approved by the master, the splitting can move ahead
b. To merge two regions, a region server sends a request to master to try to set the new merged region to merging_new, together with two daughter regions to merging. If it is ok with the master, the merge can move ahead
c. Once the splitting/merging is done, the region server reports the status back to the master either success/failure.
d. Other scenarios should be handled similarly as for region open/close

Implementation Choices

Where to Store Intermediate Region States

We used to store intermediate/transient region states in ZK. Now we need to store it somewhere else. We have two choices.

1. Store them in a system state table

Matteo suggested to store them in a new system state table. We can store all HBase system objects like tables, regions, and their states in this table. The information in this table should not be accessed by clients. Clients can access it via some master RPC calls instead.

2. Store them in the meta table

Comment [8]: Using the heartbeat? Control messages on the heartbeat again like in the old days?

Comment [9]: With hearbeat, we can do some aggregation, which is good. Only issue is that we may need to wait a little bit.

Comment [10]: In old days, as soon there was a message to deliver, we used to heartbeat; i.e. we'd not wait for the heartbeat time to elapse before passing it on... so no need to wait I'd say.

Comment [11]: What might be the conditions for approval from HM side?

Also for failures handling I assume something like that:

- if HM approves split and dies, RS proceeds and finishes split and keep attempting to report the successful split to the new HM?
- if HM approves split, RS starts doing it and dies, HM simply moves region back to non-split state, and upon next assignment the next RS will try to split it again? Does it make sense (if HM already knows region was requested to be split once) if RS fails during split, next RS can do combined recovery+split?

Comment [12]: For approval, it means the mater is not trying to move the region so as to avoid some racing. We could mandate that master to initiate the split/merge process, but the change is big. Perhaps we can do it later.

Yes, if the master is ok with the split but dies, RS can finish it. If RS dies, master should reset their states. Combined recovery+split probably is too much.

Comment [13]: Yep, combined recovery and split may be complicated, just wanted to throw it on the table so we have this option covered..

For the splits/merges to be initiated by master, we need to keep up-to-date stats about size of each region in system tables - that's probably it? Am thinking of advantages/disadvantages of having splits initiated by master - probabl ... [1]

Comment [14]: HBASE-7958 - that covers per-CF/per-region stats. Can be used for master-driven splits/merges I think?

Comment [15]: Can these be bulk operations? Bulk merge/split? Ditto w/ opens? Can we do bulk opens/closes?

Comment [16]: Only bulk open.

Comment [17]: Could these 'master RPC calls' just be your usual scan/get? We just treat them differently when target is this new table?

One concern w/ new table is whether stat ... [2]

Comment [18]: I'd assume for clients it will be read-only "dictionary" tables, so they could just use regular scans and gets, and modifications needed within the master -they can use just java locks? ... [3]

The other choice is to store region states in the meta table, some state column. We can still have the new system state table to store states for other system objects like tables, but the regions and their states are in the meta table.

The good part of this choice is that clients can still get region locations from the meta table. There is no need to break (client-side) compatibility, or put region states in two tables and make sure they are consistent.

After several releases, and all clients start to use master RPC calls to get region locations, we can simply copy the data from meta to the new system state/object table.

Upgrade Consideration

This feature will most likely be in 2.0. We mentioned that we probably won't support rolling upgrade in 2.0. However, we still need to consider how to migrate the region transition states in ZK to the meta table.

When a master starts up, currently we create the assignment znode if it is not there. With this feature, we don't need to create this znode any more. We can migrate the region transition states, and then remove this znode and its children.

If we store region states in the meta table, we can assume the state is OPEN if there is no value in the state column. If we use the new system state table, we have to copy over the data from the meta table and set the state column.

Client Change

It's better to have one system state table in the future. So we need to provide some master RPC calls for client to look up region locations.

Coprocessor or RPC

When a region is open, region server can notify master via a separate RPC call, or it can just update the meta table and master learns it from a coprocessor. The issue is that if the open failed, region server won't update meta currently. So we can't get the event from a coprocessor. The other benefit to use a RPC call is that we can make sure only master can do the updates so that we can maintain the integrity of the meta table.

Comment [19]: not sure why we should have this two things? just to have the client able to scan .META.?

Comment [20]: How about those old clients? Rewrite the scan object a little to scan regions only?

Comment [21]: the .META. scan is hidden inside HTable, so changing it to a call to Master doesn't break anyone. and I guess you have to deploy anyway hbase 2.0 on the client side to have everything working. (e.g. if you are using security you must update the client to use the new coordinated grant/revoke)

Comment [22]: That's a good point. That means 2.0 will require client app to upgrade, which may be a hard sell.

Comment [23]: client library upgrade but not code changes. we still have binary compatibility.

Comment [24]: you already broke the compatibility by changing the communication between Master and RS for the assignment

Comment [25]: You think it impossible to do this in a compatible way? Have servers support old and new systems for a release? If a master comes up and sees a region is already half-assigned using zk path, then finish it w/ zk but for new assignments go the new route? (Also, listen for zk callbacks and feed them into same code path as gets triggered when we get a control message up off the heartbeat?)

Comment [26]: we talked about this and we got a couple of solutions to make it compatible, like the coprocessor to intercept and translate. but Jon was more for avoiding tricks to keep the compatibility since we decided that 2.0 is a major and we shutdown everything

Comment [27]: Here is compatibility means client side.

Comment [28]: See above. I wonder if we can't just override the current API w/ different implementations if the special system table?

Great doc. Lets get it upstream (smile). Mikhail for one will be interested.

Comment [29]: I thought about this. If someone tries to scan meta, we use the system state table instead, which may work, not sure.

Page 3: [1] Comment [13] **Mikhail Antonov** **5/7/2014 10:55:00 PM**

Yep, combined recovery and split may be complicated, just wanted to throw it on the table so we have this option covered..

For the splits/merges to be initiated by master, we need to keep up-to-date stats about size of each region in system tables - that's probably it? Am thinking of advantages/disadvantages of having splits initiated by master - probably, more straightforward (in-process) synchronization and simpler coordination (in future)?

Page 3: [2] Comment [17] **Michael Stack** **5/7/2014 11:19:00 AM**

Could these 'master RPC calls' just be your usual scan/get? We just treat them differently when target is this new table?

One concern w/ new table is whether state will span two tables? If so, will this be an issue if they are actual HBase tables? Or, it won't be a concern because these are in-memory, local-to-the-master tables so it should be easy to do atomic modifications across them.

Advantage of new table would be could do the schema 'right'.

Would need to figure out how to do migration... or not... since we are now saying this 2.0 feature (start/stop cluster)

Page 3: [3] Comment [18] **Mikhail Antonov** **5/7/2014 6:27:00 PM**

I'd assume for clients it will be read-only "dictionary" tables, so they could just use regular scans and gets, and modifications needed within the master -they can use just java locks?

Keeping in mind multi-master setup, sounds like we could leave hookup-point to add coordination of operations between multiple masters, and already agreed operations (in case with no consensus, just all operations) could be guarded by in-process lock?