Serial replication

Background

In replication of HBase, we push Mutations to slave cluster by reading WAL in each region server. We have a queue for WAL files so we can read them in order of creation time. However, if there is region moving or RS failover, new logs in new RS may be pushed before the logs written by old RS. So we can not make sure the order of replication by now which results in some limitations:

1. If in master cluster we execute a Put and then a Delete to delete it, but push Delete first, and slave cluster does a major compaction before receiving Put, this Put will not be delete in slave cluster.
2. Replication is eventual consistency, users know they will get old data in slave cluster, but disorderd replication will make slave having some states which master cluster never have, not only old.

So we should push the Entries in order of written, more specifically, sequence id.

Brief description

Prevent pushing logs if there are any logs with smaller sequence id which are not pushed to this peer cluster, by saving some barrier/position information.

Implementation details

Add a new flag for replication peer which is called ‘serial’ to indicate that whether the entries should be pushed serially.
Add a new ‘rep_barrier’ column family in hbase:meta which is used to record two things:

1. The sequence of open sequence numbers for a region.
2. The split parent or merge parents of a region.

Store the last pushed sequence id of a region in the replication storage, currently on zookeeper.

The basic idea is, when pushing entries, we will first find out the position for the last pushed sequence id in the open sequence numbers. If we are in the first range(or even before), or the previous range has been finished, then we can start pushing, otherwise we need to wait until the above condition is satisfied.
There are special cases if a region is generated by splitting or merging. So if we are in the first range, we also need to check whether the parent region(s) have been finished.
To prevent checking meta every time, we need to do a simple optimization. We record the last pushed sequence id in an in memory map, if the next sequence id is continuous then we are safe to push.

Cleanup of stale barriers

We have a chore task to do the cleanup. The basic idea is, if all serial replication peers which contain the table for the region have been pushed beyond a given open sequence number, then all the open sequence numbers which is smaller can be removed.

Limitation and future works

It is not easy to deal with table deletion for serial replication. Replication is performed asynchronously in background. When you delete a table, it does not means that the replication for the table will be stopped immediately. So we can not delete the replication related stuffs(the open sequence numbers and last pushed sequence id) when you delete a table.
A normal way to solve this is to schedule a chore task which will wait for a safe period(maybe a day, or even a week?) before deleting the replication stuffs. But the difficulty here is that, users are free to create a table with the same name again, so it is not easy to determine whether a region is for the old table or new table(split parent or merge parents do not have data in catalog family either so it is not easy to determine this by checking catalog family).