HBASE-9465 Push entries to peer clusters serially

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 cluser 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 in hbase:meta table.

Implementation details

Add two new column families “rep_barrier” and “rep_position” in hbase:meta table to save replication information of each region.

1. rep_position:{peerid}, to save the max sequence id we have pushed for each peer.
2. rep_barrier:{seqid}, in each time a RS opens a region, it saves the max sequence id in this region. info:seqnumDuringOpen also save a seqid and it only save the latest one, but we need all.

We can keep the order of pushing from opening a region to closing it or RS crashes. The only problem is we don’t know if there are logs written before opening has not been pushed. So we use rep_position:{peerid} to save where we have pushed for each region and each peer. This record is saved after we ship the logs to peer and before we update the log position in ZK. However, sequence id is not continuous so we can not know whether a seqid larger than rep_position:{peerid} is the next log or not. So we use rep_barrier:{seqid} here. In each time we open a region, we put a barrier by max sequence id. There will be several barriers for one region. And if rep_position:{peerid} is larger than barrier_a and not larger than barrier_b, we will know there is a worker pushing logs from rep_position:{peerid} to barrier_b and all logs whose id is larger than barrier_b and not larger than barrier_c (if exist) should not be pushed to this peer until rep_position:{peerid}>=barrier_b - 1.

There are special cases: region spliting and merging. However, three related regions must be in the same region server, so the order of pushing logs from parent to daughter can be guaranteed. We need only handle region moving and failover.

This logic should be configuable because it may enlarge the delay of pushing logs when some worker waits for blocking tasks done. Set REPLICATION_SCOPE=2 to enable this feature in cf's conf. Because we have only one thread for each peer in a RS, as long as a cf's REPLICATION_SCOPE is 2, all regions' logs may be delayed but the order is not garenteed. And as long as a cf's scope is 2, all other cfs whose scope is 1 will also be serial.

Now we write region event marker in WAL, REGION_CLOSE are used in serial replication. When we read a REGION_CLOSE from a serial-scope table, we must break reading and push what we have read now. Without this, if we move a region to another RS and move it back, we may push all logs writen before moving away and after moving back, we will ignore the middle part in another RS. And we should save scope map in REGION_CLOSE. If a RS crashes, there is no REGION_CLOSE, but the region will not be moved back to this thread because the log will be read by a fail-over thread, so it is OK.

Limitation and future works

Distributed log replay is NOT compatible with serial replication. We must disable DLR.

Now we read and push logs in one RS to one peer in one thread, so if an Entry is not ready to push, all logs after it will be blocked. There is an improvment that we can read logs in one thread and use several threads and queues to push different logs in different regions. The main thread read the logs and enque them to its region's queue, so the main thread will not be blocked, and worker threads deque the logs and push them to peers. Worker threads should callback to main thread the sequence id it has pushed and save them on meta:replication, and main thread save the minimium number of the ids on ZK periodly to avoid reading pushed logs.

After this improvment, we can reduce the delay of unblocked regions and can have some more based on this change, for example, replication throttling can be in table/region level. This can be a follow-up work after the main work done.