HBase-8763 Combine MVCC and Log Change Sequence Id

Problem Statement

As of today there is no good way to know which changes of the same row are before/after another. We cannot relay MVCC as the “version” of each cell because that will cause issues when we replay/replicate WAL entries because replaying/replicating WAL entries use log sequence number order which is not in the same order of mvcc.

HBase today has a few areas where need to know the order of changes such as cell delete, same version update, compaction & out of order puts etc. In those situations, HBase puts become not idempotent. It causes those areas broken or hard to make it right.

Benefits of combining MVCC and Log Sequence Id

After combining, there is only one number & one order for each change/update. Therefore, we could use the log sequence id of each change as its version and store in current mvcc field without changing HFile format. With the log sequence id(new mvcc value) stored for each cell, HBase can make all puts as idempotent in all areas. For example, HBase cell delete can only hide changes before current delete marker not after as today.

Implementations In High Level

1) For each new update, we still firstly acquire row lock and region read lock as we do today
2) Starts a MVCC transaction with initial write number = log sequence number ++
3) Since we don’t know the real log sequence number for current change yet(which can only be known when disruptor process the WALEdit), we bump up the mvcc.writeNumber += 1000000000
   The “100000000” here is an arbitrary large number to prevent other concurrent reads to see the in-flight change and mvcc.writeNumber will be reset back later. There is a trick here that we use MutableLong(an object) to store mvcc write number. When we reset the value, KVs in memstore will immediately see the new value. Otherwise, we may advance our read point before a handler thread gets a chance to update mvcc values in memstore. For example, we have three changes in log sequence order of: edit1, edit2, edit3 while they could in order of edit3, edit2, edit1 in mvcc queue. HBase log syncer syncs edits in batch mode so it’s likely all edit1(seqNum=1), edit2(seqNum=2), edit3(seqNum=3) are synced to wal file at the same time. Therefore, the handler waiting on syncOrDefer call wakes up and advance memstore read point to 3. Since MutableLong is used, we know for sure edit1 and edit2’s log sequence number assignment happen before edit3 so does the mvcc value updates in memstore because all KVs in one mvcc transaction reference the same mvcc object.
   Adding a big number instead of using long.MAX for mvcc initial value is due to there could multiple concurrent changes for the same row and we can still maintain order as they arrive in memstore if just adding a constant value.

4) We do same logic as today: create WALEdit and call appendNoSync. For updates with SKIP_WAL, we appendNoSync a faked edit to get a log sequence as its mvcc value.
5) Before we complete current mvcc, we wait for current mvcc being assigned a real log sequence number and advance current memstore read point.

Future Work

1) Change current compaction code not to remove mvcc values from HFiles that created within 48 hrs(configurable value or just keep them forever). Therefore, all places need idem potency puts can be covered.
2) Introduce read flushed changes region recovery. Today we only allow read till recovery is complete but we could allow client to read till last flush sequence Id while the region is still under recovery because mvcc and log sequence Id are unified.
3) All other many potential improvements