Proposal to improve log splitting process regarding to HBASE-7006

<u>Problem Statement</u>

Though we have changed log splitting to distributed fashion, we still experience long running recovery when we have many WALs to replay or many regions among a few WAL files to be recovered. Currently each individual WAL replaying incurs following majors steps:

1) SplitLogWorker reads WAL once
2) SplitLogWorker writes a recovered.edits file per region per WAL
3) Region Server reads all recovered.edits files for a region before opening it
4) A cache flush for each recovered.edit file replay

IO Costs:

1) Two full scans on a WAL file: one from log splitting and the other is from replaying

2) 2 * small concurrent writes(= $\sum Wi * Ri$ where Wi represents the ith WAL file and Ri is the number of regions that Wi contains) + SUM(WALEntries contained in all WAL files)

   1) From log splitting recovered.edits file creation
   2) From memstore cache flush during replaying

The many concurrent small writes are costly because it changes underlying sequential disk write characteristics and causes extra disk seeks.

<u>Assumption</u>

edit1 in WAL1 and edit 2 in WAL2 of the same region can be replayed in different order without affecting the correctness after the full recovery is done for the region

Notes:
1) Different put order on same key/value pairs(with only timestamp difference) doesn't compromise correctness
2) A delete first and then a put on the same row key doesn't have any issue

<u>Proposed Solution</u>

Keep existing distributed log splitting framework as it is while removing individual log file splitting and instead

1) Mark regions of a failed region server as recovering marked in ZK. For example, we create znodes /hbase/recovering-regions/[encoded region name...]/[failed region server names ...]

to signal that a region isn't fully available to outside world yet because it's in recovering state. There could have sequential failed region servers when the RS receiving recovering region fails again. For each failed region server znode, last flushed sequence id of the region is stored as data of the "failed region server name" znode. (More details in section "Replay Failure Handling") We skip flushed wal edits based on the latest flushed sequence Id recorded for each region server.

2) Assign regions of failed region servers to other healthy RegionServers. Those regions will be in recovering state after opened in newly assigned region servers

3) Each SplitLogWorker reads each WAL file and sends all edits from the WAL in batch mode(grouped by region) to corresponding region servers based on assignment from step 1
Notes: we could reuse batch command with special attributes or introduce a new "replay" command. In current implementation, we use the new "replay" command option. With the new "replay" command, we are sure that traffic is from another region server not from outside clients, easily monitor and adjust "replay" priority as well.

4) After replaying a WAL file, check if all WALs of a failed region server have been successfully replayed. If it is, then remove the failed region server from ZK.

5) When no children exists under /hbase/recovering-regions/[failed region A] , remove the recovered region znode /hbase/recovering_regions/[failed region A]

6) Region server will get a nodeDelete ZooKeeper notification and free the region from recovering state.

Failure Handling

If a newly assigned region server crashes again, its managed regions will be added into recovery list by putting its WALs under corresponding /hbase/splitlog/... and appending itself into the list of /hbase/recovering-regions/[encoded region name...]/[failed region server …].
For each region per failed region server, we stores the last flushed sequence Id from the region server before it failed. The sequence Id is used to skip edits from WALs of that particular region server.
If portions of a WAL can't be replayed, existing strategies are still in use such as ".corrupt" folder and "hbase.hlog.split.skip.errors" etc.

A recovery will be only considered done when there is no region to be recovered.

Advantages:

1) All IOs(disk & network) relating to recovered.edits files creation, writing and deletion are gone for good. In addition, the proposed solution only read WAL files while existing implementation has to read recovered.edits files when receiving region servers replay recovered.edits files besides WAL file reads.

This is a huge win because HBase cluster as a whole only concurrently opens normal WAL files(not counting hfiles). Therefore, the performance won't degrade no matter how many WAL files to be recovered because the number of concurrently opened files is determined by normal operations not affected by recovering logic. HBase is write efficient so just dumping data is a piece of cake.

2) Allow writes(puts) from outside clients even during a region recovery phase to increase availability. This is useful for time series data like usage to avoid data loss when a region is in recovery.

3) Cleaner code: potentially remove all special log splitting code and rely on existing common write path for recovery operations.