HBASE-26323 Introduce a SnapshotProcedure

Background

Currently, the snapshot in hbase uses zk as coordinator. It has some limitations,

a. Snapshot may fail when there are region servers crashing.

b. Snapshot may fail when master restarts.

c. Only one snapshot per table can be taken at the same time.

d. Snapshot verification will be handled by the master, which may take a long time when our table has a large number of regions, for example 10000.

Since we have procedure v2 framework now, it is possible to solve the above problems.

Goals

a. Snapshot can continue when there are region servers crashing.

b. Snapshot can continue when master restarts.

c. More than one snapshot per table can be taken at the same time.

d. We can use RegionServers to verify the snapshot to accelerate the procedure.

Implementation

SnapshotProcedure will be a StateMachineTableProcedure.

About concurrency control

Snapshot procedure can't acquire exclusive table lock because

a. exclusive lock has a negative effect on assigning regions. See HBASE-21480 for details.

b. We want to support taking multiple different snapshots on the same table at the same time.

So the snapshot procedure should acquire a shared table lock. but this is not enough. The split/merge procedures also acquire shared table lock and region lock. Snapshot procedure should also check if there are already split/merge procedures running, snapshot procedure should wait split/merge procedure to finish (The split/merge procedures will always check if there are snapshot procedures running before execution).

I have some check solutions now, but I am not sure they will work, just list them.

Solution A:

```
private void prepareSnapshot(MasterProcedureEnv env)
      throws IOException,ProcedureSuspendedException {
    if (thereAreSplitOrMergeProcRunning) {
      setTimeout(Math.toIntExact(backoff));
      setState(ProcedureProtos.ProcedureState.WAITING_TIMEOUT);
      skipPersistence();
      throw new ProcedureSuspendedException();
    }
   ……..
}
@Override
protected synchronized boolean setTimeoutFailure(MasterProcedureEnv env) {
    setState(ProcedureState.RUNNABLE);
    env.getProcedureScheduler().addFront(this);
    return false;
}
```

Solution B:

```
// this may have race condition
private void prepareSnapshot(MasterProcedureEnv env)
      throws IOException, ProcedureSuspendedException {
```

```
    if (thereAreSplitOrMergeProcRunning) {
      ProcedureEvent<?> event = new ProcedureEvent<>(this);
      AtomicInteger counter = new AtomicInteger(numRunningSplitOrMergeProc);
      for (long procId : runningSplitOrMergeProcs) {
        env.getMasterServices().getMasterProcedureExecutor().registerListener(
          new ProcedureExecutor.ProcedureExecutorListener() {
          @Override public void procedureLoaded(long procId) { }
          @Override public void procedureAdded(long procId) { }
          @Override public void procedureFinished(long procId) {
            if (counter.decrementAndGet() == 0) {
              event.wake(env.getProcedureScheduler());
            }
          }
        });
      }
      event.suspendIfNotReady(this);
      throw new ProcedureSuspendedException();
    }
 …..
}
```

Solution C:

```
// this may stuck worker thread of procedure executor
private void prepareSnapshot(MasterProcedureEnv env) throws IOException {
  // ......
  if (thereAreSplitOrMergeProcedureRunning) {
    CountDownLatch latch = new CountDownLatch(numRunningSplitOrMergeProcedure);
    for (long procId : runningSplitOrMergeProcedureIds) {
      env.getMasterServices().getMasterProcedureExecutor().registerListener(
        new ProcedureExecutor.ProcedureExecutorListener() {
        @Override public void procedureLoaded(long procId) { }
        @Override public void procedureAdded(long procId) { }
```

```
        @Override public void procedureFinished(long procId) {
          latch.countDown();
        }
      });
    }
     // this may stuck worker thread of procedure executor
    latch.await();
  }
}
```

Maybe there are better ways, I am not sure.

About state

snapshot procedure will have following states:

SNAPSHOT_PREPARE = 1;

initialize some runtime variable like working dir

SNAPSHOT_PRE_OPERATION = 2;

create or reset working dir. call preCorprocessor.

SNAPSHOT_WRITE_SNAPSHOT_INFO = 3;

write snapshot info to hdfs.

SNAPSHOT_SNAPSHOT_ONLINE_REGIONS = 4;

snapshot online regions. This will be implemented with a remote procedure.

SNAPSHOT_SNAPSHOT_OFFLINE_REGIONS = 5;

snapshot disabled table regions and split parent regions. This work will be handled by the master. For disabled table, even if the number of regions is large and snapshot is slow, it will not hurt the availability, and since we have proc-v2 client now, there will not be TimeoutException. For split parent regions, in most cases, the number will not be large, so I think it's ok to let the master do this work.

SNAPSHOT_SNAPSHOT_MOB_REGION = 6;

snapshot mob region. The master will do this work too.

SNAPSHOT_CONSOLIDATE_SNAPSHOT = 7;

read all region manifest and write into a single snapshot manifest file.

SNAPSHOT_VERIFIER_SNAPSHOT = 8;

check the consistency of snapshot and real data. This will be implemented with a remote procedure.

SNAPSHOT_COMPLETE_SNAPSHOT = 9;

commit snapshot manifest file, update metrics and monitored-task status

SNAPSHOT_POST_OPERATION = 10;

call postCorprocessor.

About rollback

just delete the working dir.

About SnapshotManager

The snapshot manager needs some extra work to cooperate with the snapshot procedure.

work A. keep track snapshot procedure

In many cases, SnapshotManager needs to know whether there are snapshots in progress. Here are some examples.

1. when client submit a new snapshot request (currently we have limitation that the name of different snapshot can not be same)
2. When split/merge procedures start to execute, they will ask the snapshot manager if there are snapshots in progress. if there are, split/merge procedures will fail.
3. when HFileCleaner starts to execute, it will ask the snapshot manager if there are any snapshots in progress also. if there are, HFileCleaner will not run.

SnapshotManager needs to know if there are snapshot procedures running. A possible way is to check all the procedures in the procedure executor, but this can be expensive. For zk-coordinated snapshot, we have a map to record the table and SnapshotSentinel. maybe we can use a map to track snapshot description and snapshot procedure like this too.

```
// SnapshotDescription -> SnapshotProcId
private final Map<SnapshotDescription, Long> snapshotToProcIdMap = new HashMap<>();
```

This map will help snapshot manager search snapshot manager. The finished procId will be cleaned by the SnapshotSentinelCleaner.

work B. rebuild the above map when master starts

When the master restarts, the above map should be rebuilt. A possible way is to let the snapshot procedure register itself after replay. However the procedure executor starts before the snapshot manager, **so the snapshot manager has not been initialized when we replay procedures.** So we may need to check all procedures in the procedure executor when initializing the snapshot manager.

work C. keep the working directory of unfinished snapshot procedure

When master starts, it will delete and recreate the global working snapshot directory (/<root dir>/.hbase-snapshot/.tmp). It's ok for zk-coordinated snapshot because they will fail if master restarts. But the snapshot procedure can continue. So we should just delete the zk-coordinated snapshot working directory.

About the communication between parent procedure and child procedure

Parent procedure must have a way to check if child procedure really succeeded or not and have appropriate handling. I think this is the most difficult part to introduce a new procedure with child procedures. According to my understanding, a procedure should only

communicate with procedure env (scheduler, executor, store). Procedure should not communicate with another procedure. (maybe my understanding is wrong)

There are two common ways of communication

1. based on hdfs or zk. For example, SplitWALProcedure checks if SplitWALRemoteProcedure really succeeded or not by the wal path. if path still exists, SplitWALProcedure will retry.
2. parent procedure just ignores the result of child procedure execution. for example, SwitchRpcThrottleProcedure and SwitchRpcThrottleRemoteProcedure.

The snapshot procedure has two kinds of child procedures, SnapshotRegionProcedure and SnapshotVerifyProcedure.

For SnapshotRegionProcedure, we will keep retrying until we succeed.

For SnapshotVerifyProcedure, if it doesn't know if the snapshot is corrupt or not, it will keep retrying. If it knows, how can it tell the parent procedure the result ? we can't use snapshot manger because master can be restarted at any time, that may cause result loss. So I think a possible way is that if the snapshot is corrupted, mark the snapshot verify procedure as failed, that will trigger the rollback of the whole snapshot procedure stack. I am not sure if there are better ways.

As I mentioned above, communication between parent procedure and child procedure is very difficult. So we should avoid communication. for ServerRemoteProcedure, if dispatch fails, it just returns and lets the parent procedure find out and retry. Different from ServerRemoteProcedure, we have to use some tricks to handle FailedDispatchException, so we have to override the execute method to add our logic.

For SnapshotRegionProcedure, if dispatch fails, it will suspend for some time and retry.

For SnapshotVerifyProcedure, if dispatch fails, it will yield and retry on a new server.

About introduction

The simplest way to introduce to the client I can think of is by introducing a new method snapshotTable. If there are code bugs in the snapshot procedure and can't finish the procedure, we can still use the zk-coordinated snapshot.

Maybe we can introduce the snapshot procedure in 3.x, make this default in 4.x, deprecate zk-coordinated snapshot in 5.x, and finally remove zk-coordinated snapshot in 6.x. I am not sure. Let's make this decided by the community.

About sub tasks

1. abstract some sanity check logic in the snapshot manager for both zk-coordinated snapshot and snapshot procedure
2. abstract snapshot verify logic to a util class
3. introduce a snapshot verify procedure
4. introduce a snapshot region procedure
5. introduce a snapshot procedure
6. introduce a snapshotTable method to master (snapshot manager)
7. introduce a snapshotTable method to client
8. add shell support for snapshot procedure