Apache HBase - Direct insert HFiles and Persist in-memory HFile tracking

Tyler Jan, Abhishek Khanna, Stephen Wu, Zach York (Alphabetical Order)

Problem

For users that have been running HBase on S3 object stores, the commit stage in flush and compaction that renames/moves HFiles from .tmp directory to data directory is expensive and slow. Behind the scenes, renaming a file requires a full-copy (+delete) on the object stores instead of a constant time metadata operation that HBase expects. We want to revisit the commit logic and directory structure within HBase to remove rename as a commit operation.

With the introduction of storefile management in HBASE-7603, HBase uses a StoreFileManager to keep an in-memory cache of the HFiles that are presented in the system. This StoreFileManager is used to determine which files are available for operations such as scan, compaction, split without having to list the filesystem each time. Utilizing the StoreFileManager, HBase can be improved to provide a simpler directory structure and optimize the expensive commit stage for object store filesystems.

We propose removing the `.tmp` directory used in the commit stage for common operations such as flush and compaction to improve the write throughput and latency on object stores. Specifically for S3 filesystems, this will also mitigate read-after-write inconsistencies caused by immediate HFiles validation after moving the HFile(s) to data directory.

Technical Approach

The initial prototype was targeted for flush and compaction, but we will extend this to other operations that utilize the temporary directory for a commit mechanism

Introduce a new Store Engine and direct insert HFiles to data directory

We will introduce a new store file engine, store flusher, store file compactor, and store committer that flush and compact HFiles directly into the data directory. Here, a store committer is generally responsible for committing the flushed HFiles to data

directory and completes with updating the last available HFiles tracked in store file manager.

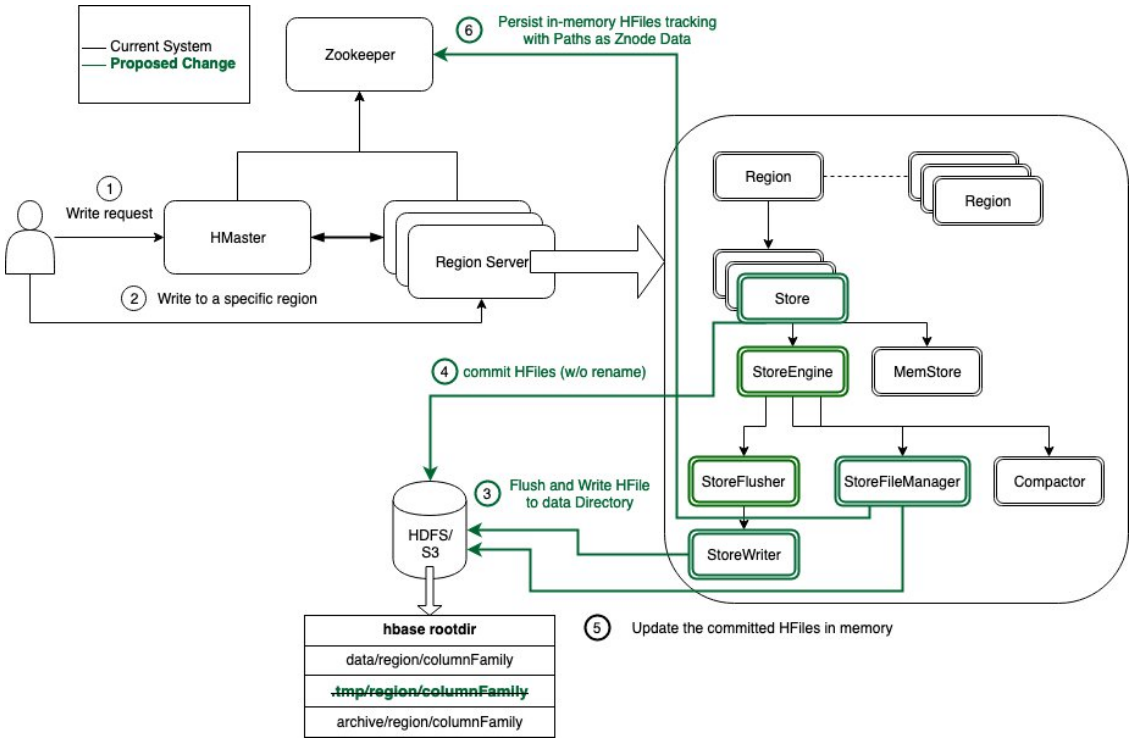
Introduce Persistent StoreFileManager

For recovery from region server crashes or region reload, we persist the in-memory HFiles tracking in store file manager to a new HBase admin table, 'hbase:storefile'. To prevent loading HFiles from incomplete flushes and compactions, and reduce the number of expensive LIST files calls against the file system, we will read directly from the hbase:storefile table. A write to the storefile table is used as the commit mechanism for a HFile, removing the rename from .tmp to the data directory. To avoid a circular dependency on the storefile table, the store file manager for the meta and storefile tables will be persisted in ZooKeeper.

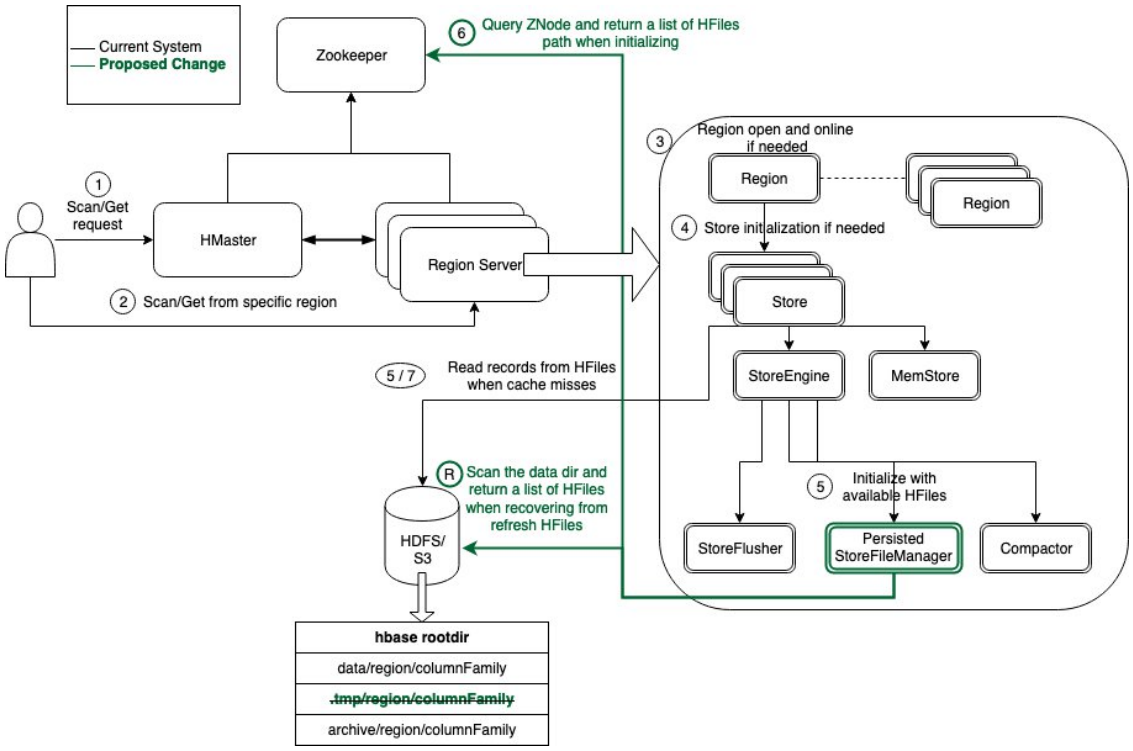
To clean up uncommitted HFiles left in the data directory, we will introduce a periodic cleaner to move the incomplete files from the data directory to the archive directory. Additionally, if a user wants to ensure the data directory is clean on region open, a new feature could be added to delete unnecessary files during the region open procedure.

High-level overview

Flush or Compact



Store Opens



Upgrade considerations

This new feature is optionally enabled, and the upgrade with or without this feature is very simple. By default the feature is off, the commit stage with rename and HFiles available to this store are exercised as usual . When this feature is turned on for the first time, we will build the hbase:storefile table using the actual files available to each store on disk.

We're proposing to add a new component, a StoreCommitter, to StoreFlusher or StoreEngine but since these API changes are private and internal only to hbase-server, it should be safe for user to upgrade between versions. However, we conceptually change the view of data and the directory structure. If any current HBase operators have been writing HFiles to the data directory without going thru the Store Engine, one may need to run refresh hfiles operation (RefreshHFilesClient) to update the view of available HFiles after using this feature in those uncovered use cases outside of normal HFiles writes via HBase client.

Testing

All functionality will be tested with unit tests, mini HBase cluster tests and performance tests on S3. Tests would validate all HFiles read and write operations (e.g. flush, compact, split, merge, snapshot etc) that have the assumption of .tmp

directory. Cluster integration testing to ensure all the desired behavior in regard of using HDFS implementation, would also be carried out.

For performance tests, due to the removal of rename in the commit phases, we will execute YCSB benchmark tests with and without our approach by comparing among implementations on S3 object stores, e.g. pure S3A with this change vs pure S3A vs S3Guard, EmrFs with this change vs pure EmrFs vs EmrFs consistency view

Documentation

It will be very important to document this feature, especially after users turn on the feature and if they have any hooks to insert Files to the data directory outside of HBase routine. So, we're planning to add a new section to official guideline describing this change, how the data directory has changed and can now temporarily have uncommitted data. The documentation will also include instructions on how to rebuild the HFile tracking or migrate from a cluster without this change.

Benefits with this proposal

1. Lower write latency , especially the p99+
2. Higher write throughput on flush and compaction
3. Lower MTTR on region (re)open or assignment
4. Remove consistent check dependencies (e.g. DynamoDB) on the file system

Related approach

- For direct inset into data directory and rename removal, HIVE community, HIVE-21164 also accepted a similar idea to avoid a move step during inserts/compaction by making the Compactor run in a transaction HIVE-20823.
- For s3 object consistency guarantee with current HBase's required file operation semantics, HBoss in HBASE-22149 provided a locking methodology to make sure files are available after rename with the help of S3Guard.

Reference

1. https://issues.apache.org/jira/browse/HIVE-21164
2. https://issues.apache.org/jira/browse/HIVE-20823
3. HBOSS https://issues.apache.org/jira/browse/HBASE-22149