FS Layout Redo Sketch

Comes of running chat between Matteo, Srikanth, Appy, and St.Ack, 20150714

Host JIRA: HBASE-14090 *Redo FS layout; let go of tables/regions/stores directory hierarchy...*
Forerunners: HBASE-7806 *Isolate the FileSystem calls*

Our layout will not allow us scale (*too-many-files-in-one-directory*). It has other issues in that a consistent view on tables and snapshots is complicated by ownership being split between Master and DFS. File cloning, references, renames, and garbage collection are expensive (NN RPC), tenuous operations each with its own unique implementation.

This sketch breaks physical layout purging notions of table/region/store from DFS. Instead, these constructs are owned solely by Master.

[1]1 Why

- Scale to 1M+ regions
- Remove small files (Split Reference, HFileLink), our homebrewed ‘linking’
- Proper (and less expensive) atomicity by avoiding create /tmp && rename
- No more *catch FileNotFound exception* and then look for file in alternate location (HFileLink)
- Better S3 support; FS rename is a copy in S3 (Try to compact a 50G file and rename it)
- Online Snapshot w/o need of flush; no need for RSs coordination
- HFile Cleaner without fs scan, can even be event based
- Table rename

Our table/regions/column family 'facade' has to be maintained in two locations – in master memory and in the hdfs directory layout – and the ‘farce’ needs to be kept synced or worse, the model management is split between master memory and DFS layout. This distribution makes it hard to make changes and to enforce integrity.

2 Sketch

Let us do a sketch first. Can then drill down to add detail in later sections. TODO: Pictures

hbase:meta has a row per region with *hregioninfo*, server the region is being served on, as it has now, etc. Add a column with the list of files that make up a region by columnfamily to this *hbase:meta* table region row[2] On flush and compaction, manipulate this set atomically up in hbase:meta.

In HDFS, add a new data dir. All data files (hfiles) are added to this data dir with no grouping by table and then by region, and so on. Instead make tiers of subdirs just to avoid the *too-many-files-per-dir* issue (Scheme TBD). What files comprise a columnfamily and then region is found by reading hbase:meta, not a directory layout in HDFS. Ditto for tables.

3 Downsides

- Only had to have HDFS up to ‘see’ HBase data; useful debugging
    - Counter: Reading HDFS is NOT a proper view on HBase data.
- Could use HDFS quotas on directories to do HBase quotas
    - Counter: Starts to breakdown when files are shared between tables/dirs; e.g. snapshotting
- You must have an HBase cluster running to access HBase data; no more backdoor

4 Detail

---

[1] A sketch done by Matteo, a main input to this doc

[2] "Each tablet’s SSTables are registered in the METADATA table." Bigtable Paper

4.1 hbase:meta

Rows have new columns of columnfamily to files.
On region open, Master passes a *manifest* of table descriptor, region name and descriptor, and files that make up the stores/column families. Master gets the manifest from a reading of hbase:meta.

hbase:meta will now carry more load. Critical that hbase:meta can be split and distributed[3].

4.2 Data Directory

Dump all files into a data dir in HDFS. Make subdirs under this data dir follow a scheme perhaps like that of the DataNode, where we shave the first two characters off the front of the block filename to make the first subdirectories, then characters 3 to 4 for the next level of subdirectories, and so on. For example: */hbase/data/abc/def/ghi/abcdefghijklmnopqrstuvwxyz*

As in the datanode, we just need to pass the filename of the HFile since its location can be derived (at least in the above naming schema this is the case). Same in *hbase:meta*; we'd just need to store the filename; no need of path.

4.2.1 Permissions

"How do we manage permissions? At the storefile level? What if we have to change those permissions? 3.a. We need to give users 'x' access to all the data dirs. 3.b. Secure bulk load changes storefile permissions to 777 because of chown limitation. With this approach a user can potentially access a storefile he/she not supposed to." Francis Liu

4.2.2 Pros/Cons

For Francis Liu

4.2.2.1 Pros

-

4.2.2.2 Cons

- Lose tie of logical and physical grouping of tables/regions/stores
    - Cannot use hdfs permissions to enforce hbase dir access
- If hbase:meta is corrupt, recovery is impossible. If corrupt, has potential to amplify dataloss (a missed edit might mean a missed hfile)
    - Counter: recovering meta from a read of hdfs was a fuzzy, last resort not exercised in eons.

[3] If we only have to keep filenames and not full paths and if filename is current 32 character then each region row given say 10 Column Families and ten files each would be 10 * 10 * 32 = 3.2k extra per row. At 1M regions is 3.2G or so.

4.3 Files

Flushes create files. Flushes would now have to update the *hbase:meta* with the newly added file entry. Will have to update meta with files written on close of region even when cluster shutdown: i.e. hbase:meta would have to be last region to go down so can take the twilight writes of all other cluster closing regions. This is currently the case but would need to be even more so now.

Flushes are already noted in the WAL. On WAL replay, include replay of the insert of the file into hbase:meta step. Ensure idempotent.

Keep up a reference count on each file in hbase:meta so we know when a file is on longer referenced and can therefore be garbage collected. If done inside the region row in hbase:meta, a file cleaner would have to do a full table scan each time it ran to find files that had reference count of zero.

An optimization would add a row per file to the hbase:meta table; i.e. the hbase:meta table would be in two halves with one half, a row per region as we have now and then, in the second half, a row per file. We'd add to the hbase:meta rather than create a new special system table so we keep our list of *special* tables to a minimum thereby keeping assign simple. Reference counting would be done on the file row. The file cleaner would scan this file subset of the the hbase:meta table only. Or better still, an hbase:meta coprocessor could react when reference count reaches zero and garbage collect the file immediately; no need of a scan.

Over a files lifecycle, it could be referenced by two regions (split), be dropped from a region on compaction, etc., but it will not be removed until its reference count is zero. This new file row section would not have to be atomically sync'd with the reference in the region row as say you would do for a secondary index. The add of the file row could be done lazily on say flush of the hbase:meta; on flush add the file row and update reference count (if crash, replay of WAL would redo this step).

4.3.1 References/Range

Currently, when we split a region, we create new Reference files in the daughters, one per file in Parent. Reference file is a pointer to parent file and whether we are to read TOP or BOTTOM half of the parent file in the daughter context.

In new layout, one approach would have no need of a Reference file in the filesystem. Just add attribute in hbase:meta on file in daughter region stating what part of file the daughter is reading (TOP/BOTTOM, etc) and on region open, on referenced hfile open, it has all the metadata it needs to do an HalfFileReader.

TODO: How we update refcount on files referenced by daughters in atomic way (and decrement parent reference).

TODO: Enis suggests we might allow finer grained ranged that TOP and BOTTOM: i.e. split files in quarters, eights, etc.

4.3.2 Links

HFileLink is used so can continue to reference a file though it may have been moved. Used mostly by snapshots to insulate against file being moved to archive dir when no longer needed by serviing region: e.g. it has been compacted away.

In new regime files will not be moving. Per snapshot, just up the refcount on the file.

4.4 Snapshots

4.4.1 Snapshots without Flush, without RS coordination

4.4.2 MapReduce over SnapShots

4.4.3 Table View is a 'snapshot'

From Matteo: if accidental delete, can restore table if we keep tables around with a TTL before we start removing their content.

4.5 Integrity/HBaseFSCK

4.6 Online Migration

We may be deluded but we believe a rolling restart up on to new scheme is doable.

Basically, start new master. Do a rolling restart. On assign, if N new regionservers are online (should be > 1), then assign with *manifest* which will trigger the regionserver to migrate region to new scheme (move files to new location, etc.)

4.7 Prerequisites

None

5 Questions

5.1 Why not just have HBase manage raw blocks altogether?

See exchange that starts here up in hosting JIRA.

5.2 Tooling

5.2.1 “What HFiles are in a single store? What does their modification times tell me about flush and compaction behavior? Are there too many or too few, relative sizes of column families? Can we replace this with good meta analysis tools?” [Dave Latham]