Restore Snapshot: Hardlink alternatives

Taking a snapshot means creating a “reference” to the current HFiles + Keeping the HLog + Dumping (META) regioninfo.

- HFiles can be removed due to a compaction.
- HFiles can be removed due to a region deleted.

Taking a Snapshot

As described in the in HBASE-6055 a snapshot is a “clone” of the table directory but instead of having real files you’ve a reference to them.

- /hbase/<table>/<region>/<cf>/<hfiles>
- /hbase/.snapshots/<snapshot name>/<region>/<cf>/<hfiles>

HLog and “meta” information are special cases and will not be covered in this doc.

Hardlink support

Supposing for a moment that we’ve hardlink support (HDFS-3370).

- “Take a snapshot” is just a matter of creating a reference to hfiles
    - This is just a call to fs.link() for hfile in the table directory.
- No needs to change the code to handle file removal, hardlink refcount does the magic.
    - the file will be removed just when refcount is 0

As you may see, with hardlink support taking a snapshot is an easy operation, from a 1000ft can be described as “for each hfile in the table directory call link()”. No code changes required to handle file deletion during compaction or region removal (drop table).

Restoring a Snapshot

We can have two types of restore:

- Restore snapshot - is the operation that allows to rollback the current table to a “previous” state (snapshot).
- Mount snapshot - is the operation to load the snapshot in a new table to be able to look at snapshot side-by-side with the current table. (This allow to run MR job to copy just a portion of the table, corrupted rows, or just to show the differences between the two version, …)

In both cases the operations are quite similar, and the main idea is to replace the current table directory content with the ./snapshot/<snapshot name> directory content (1000ft description).

Again, with the hardlink snapshot this operation is super easy: “for each hfile in the snapshot directory call link()”. Without any modification hbase read those hfiles and can remove them without problems hardlink refcount does the magic!
Unfortunately we don’t have hardlink support, so we need an alternative.

And the main problem is that we need some way to:

- create a reference to the hfiles
- move the hfiles that we want to delete to an “archive” directory (if hfiles are referenced)
    - update the references

Reference Files

The current Jesse's implementation HBASE-6055 creates a Reference file that "points to" the original hfile. When the hfile need to be removed due to a compaction/region deletion it will be moved to an .archive directory.

This is great because we know that if the hfile is not in the original directory is in the .archive, and we can implement the "Restore snapshot" functionality without problems. If the file is in the original directory we've nothing to do, otherwise we need to move back the archived file.

But what about the "Mount snapshot" functionality?
The simplest solution here is to do physical copy of the hfiles to the new table. But this operation cost lots of time especially if for large tables.

Another solution is to copy the reference files instead of the real hfiles, this allows to create the table super quickly. But what happens when a file is deleted from the original table, or is moved back from the archived files to the original table? In this case we need a try … catch FileNotFound look in archive or original.

Also refcount is not free, since we don't know how many references we've around (multiple snapshots for the same table, multiple mount, …). In this case we don't have a manual cleanup when we reach refcount 0 (we don't have refcount at all), but this is easily solvable with a cleanup tool that scan each reference file and keep tracks of what is cleanable.

Keep refcount in .META.

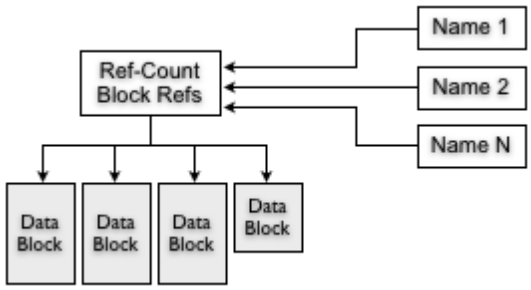
This solution was proposed in HBASE-6205 and is sort of what Accumulo does.
In .META. you add another column family to keep track of files used by each table.

This allows to keep track of number of references of one file, but doesn't solve the file move problem that we've with the reference files. When a file is deleted due to a compaction/region deletion we need to move that file somewhere and update all the references.

Also having lots of file can slow down the .META. operations.
The only advantage of this approach is that we've a way to know which files are referenced and we can delete them when are no longer needed.

Move & Symlink

Taking a step back... What is an Hardlink? What can be a simple implementation?



You can think to an hardlink as a extra name layer on top of what we have. This means that you've some sort of Hidden I-Node that contains refcount and blocks pointer. And each "real" I-Node point to that one.

How can we simulate an Hardlink?
Since at some point we need to move the hfiles to the "archive" directory due to a deletion, we can move the hfiles during the snapshot phase, and replace the original hfiles with a symlink to the

archived hfiles. In this way we've something similar to an hardlink. Removing a symlink doesn't effect the original file. The only problem is that we don't have the refcount. But we can have a cleanup "tool" as the other alternatives (Reference Files, .META refcount).

Trying to go through the various operations:

- Take a snapshot - is consist of:
  - Move the hfile to archive
  - Create a symlink to point to the archived file
  - Create a symlink for the snapshot
- Restore snapshot - creates the symlink to the snapshot symlink (that point to the archived file).
- Mount table - creates a new table with symlink to the snapshot symlinks.

From a filesystem point of view what we've is something like this

- /hbase/.archived/<hfile>
- /hbase/.snapshot/<snapshot name>/<region>/<cf>/<hfile> -> /hbase/.archived/<hfile>
- /hbase/<table>/<region>/<cf>/<hfile> -> /hbase/.archived/<hfile>
- /hbase/<table 2>/<region>/<cf>/<hfile> -> /hbase/.snapshot/<snapshot name>/region/<cf>/<hfile>

Conclusion

The last solution "Move & Symlink" allow to keep the hbase code intact as the solution with the hardlink support while the other solutions require more code and changed to handle delete situations.
Also in a future transition to hardlink if we want to keep the compatibility with the snapshot taken, we need to keep the code needed to read the references or the extra .META. column while the the last solution we don't have to care about nothing since there are no changes required to what we've now in hbase.