The S3A directory marker problem

HADOOP-13230

Steve Loughran, February 2020

https://issues.apache.org/jira/browse/HADOOP-13230

Summary

The effort put into deleting fake directory markers slows down S3A Filesystem operations, generates excessive load on S3 partition and so may trigger throttling across an application, and, because it creates tombstone markers, and slow down operations against versioned S3 buckets.

We can't stop creating them because applications which expected newly created directories to exist will break. This includes Hive, spark, etc.

We can stop deleting them -but need to do so carefully so as to avoid breaking all previous versions of the S3A client, versions which interpret them as empty directories for operations such as rename and delete.

History
/* TODO, insert history for the markers in general */

Proposed

1. We change probes for directories such that the presence of a directory marker is never interpreted as representing an empty directory, simply "a directory which may or may not have children".

2. We add an option to stop deleting directory markers after files and directories are created, e.g `"fs.s3a.keep.directory.markers"`
The options will be:

- `true`: keep them. The new option
- `false`: delete. Backwards compatible. *This must be the default*
- `authoritative`: an intermediate step -only keep them on authoritative directory trees, on the expectation that specific releases of Hadoop will be working with those directories.

3. We provide patches for all hadoop branch-3 releases which stop interpreting a directory marker as an empty directory. The code to eliminate marker creation will not be back ported -this patch is to guarantee compatibility with later releases which do retain markers.

4. We identify and tune those operations which probe and list directories so the presence of directory markers will not slow them down.

S3A Directory Markers: the reason

1. A lot of applications have an expectation that newly created directories are visible through the hadoop filesystem API
2. To offer this behaviour on s3a (as its predecessor s3n did), we add directory markers: 0-byte objects with a trailing /
3. For file existence, listings etc we do a HEAD request for the markers.
4. Currently the connector assumes that the presence of a marker means that there is an empty directory at that point - a fact used in those operations which explicitly change their behaviour based on this state (rename, delete non-recursive)
5. To ensure this condition holds, FS operations which create files or directories (including through renames) need to delete or parent markers. Rather than scan and delete, we just issue a single bulk DeleteObjects call listing all parent marker paths. This is much faster.
6. `mkdirs()`walks up the tree to verify that the first extant parent is a directory.
7. Note: we don't do any of these checks on file creation - it is expensive and other than in contrived tests, nobody ever tries to create files under files.
8. Directory renames and deletes build up lists of objects and then issue aggregate DeleteObject requests in blocks (historically pages of 1000 - this being the maximum size; now down to 250)
9. When a directory or file is deleted we have to probe the parent for being an extant directory - if not a new marker is created. Again, code expects that after `rm(path)`, `isDirectory(path.getParent)` is still true.

The consequences

- Probes for files and directories have to add a HEAD path + "/" to identify these markers. As this costs slightly less than a LIST call, we do this before a LIST for children - but whenever working with non empty directories it adds the the delays and cost of a needless check (one which adds a 404 to the S3 cache, incidentally)
- Every delete call adds a new tombstone in a versioned S3 bucket. this will slow down at subsequent operations against paths in/above these paths, Lists in particular. Temporary directories are apparently a key problem here because so many files are created and then deleted.
- Directory renames and deletes of many thousands of files can trigger throttling on a specific shard of an S3 bucket - and as a consequence slowing down all operations against the shard - especially active hive queries.
- When the account/role of a client lacks write permissions up the directory tree For a destination of a directory creation file creation or rename, the delete will fail. These

errors are swallowed - but they do mean directory markers can end up being retained above files and directories. That is something our current code cannot handle well.

Hive is a particular troublespot here because it does not use a high-performance S3 a specific committer. It creates and renames many directories, and its use of the directory based partitioning of data results in deep directory trees.

Spark is slightly less disadvantaged because the S3A committers avoid directory renaming, but it still relies on recursive directory listing to identify source files - the presence of (fewer but still non-zero) tombstone markers will affect query planning. It does at least avoid rename/directory-triggered throttling.

History

HADOOP-13164 Optimize S3AFileSystem::deleteUnnecessaryFakeDirectories,
HADOOP-14255 S3A to delete unnecessary fake directory objects in mkdirs()
HADOOP-14428 s3a: mkdir appears to be broken
HADOOP-13221 s3a create() doesn't check for an ancestor path being a file
HADOOP-16823 Large DeleteObject requests are their own Thundering Herd

What can we do?

stop deleting markers as files and directories are created.

Thus: no need to issue delete requests after creating directories or files. And so: no S3 load generated by the deletes, and no tombstones in versioned stores.

This is going to leave those markers around -it is okay then to be interpreted as directories (it is true, after all) -but their presence must not be interpreted as "this is an empty directory", which is precisely how they are treated today. Instead of those operations which do ask for empty directories called in rename() and delete() must depend on a real list taking place.

We should be able to fix that up in `innerGetFileStatus(path, needEmptyDirectoryFlag=true, Set<Probes>)`

After which: we should all be transparent to all releases with the same modification.

It will however break every shipping release, including every application which has expectations about rename() moving the content of directories -which includes Hive, amongst others.

That is: older applications will be unable to safely interact with S3 buckets where a more recent release of the S3A connector has stopped deleting directory markers. That includes Hive, Spark, and DistCp, to name but a few

Compatibility

When we stop deleting markers we will break all previous hadoop releases, especially their `rename()` and `delete()` operations.

`delete(dir, recursive = false)`
This will not fail (it's an empty dir). The actual deletion of descendant files may or may not take place

`rename(source, dest)`
  - if there is a marker at `source`, the directory will probably be considered empty and so no child files copied.
  - if there is a marker at `dest`, it will be interpreted as an empty directory and so all source children files merged under `dest` alongside any which exist. That is

```
children(dest') = children(dest) + children(src)
```

This will make it impossible for the older releases to work with S3 buckets which have been written to by an updated version.

We are going to have to make this feature optional.

And even there, provide a patch for all supported releases, as options notwithstanding, older applications will end up being targeted at a bucket whose previous clients left directory markers. ASF releases of other applications (Hive, Spark, etc) are a particular risk here -and because they tend to stick to older point releases, are not going to rush to upgrade to "Hadoop 3.4.0" binaries. We will have to issue 3.1.x, 3.2.x and 3.3.x minor releases and encourage their adoption.

This will complicate testing, those that we will have to somehow write a compatibility test which verifies that current shipping releases are compatible with work performed by the optimised version in buckets/paths where directory markers are not retained. For due diligence, that means against the real binaries rather than as having a switch to reinterpret markers as before.

Related issues

Can we stop having to probe for and recreate parent directories as the files/directories are deleted or renamed?

No

We need to do that to preserve the metaphor which applications like Hive expect: delete something and its parent dir is still there.

Stop applications failing when parent directories of deleted directories no longer exist and maybe we can worry about not creating fake dirs.

Assuming that many fake markers will eventually accrue across a bucket, it may be more efficient to retain the current (HEAD + /, LIST) ordering, rather than the inverted ordering planned. However, directory trees created in bulk operations (rename, commit) won't have markers, so the benefits are likely to depend on the origin of datasets -in particular the commit algorithms used.

Put differently: "if your workflow is still renaming many gigabytes of data -worry about that and not to the overhead of some probes up the tree after the rename"

Changes to S3A Operations

Path probes

`innerGetFileStatus(Path, needEmptyDirectoryFlag, Set<StatusProbeEnum>)`

This is where we need to change the needEmptyDirectoryFlag check to actually list the children rather than interpret a directory marker as an empty directory

This is the change which requires the most care, and is the one which we will have to backport everywhere

One thing we have to worry about is the fact that the LIST path + / operation may now return the directory marker, which will then be interpreted as "this path represents a non-empty directory"

We need to recognise this and differentiate "listing found marker" from "listing found at least one child". That may require making a larger limit (i.e > 1).

Note: this bit of probing has been trouble in the past where in the list finds an object under the path -but it is one for which S3Guard has a tombstone because it has been recently deleted. (see HADOOP-16380 and other JIRAs related to `ITestS3GuardDDBRootOperations`)

`getFileStatus(), isDirectory(), exists()`

Reorder to look for a LIST entry ahead of the marker.

1. `getFileStatus() & exists()`: HEAD, LIST, HEAD + /
2. `isDirectory()`: LIST, HEAD + /

These checks are potentially suboptimal if you leave markers around, but given directory trees created by Hive's renaming and spark S3A committers will not be created with markers, it should be faster against real datasets.

Listing

`listStatus(), listFiles(), listLocatedStatus()`

Proposed order : LIST; HEAD + /; HEAD

The directory listing operations currently all do a getFileStatus() call, of HEAD, HEAD + / and LIST before the actual list of the contents. This is slow and expensive, especially given all application tree walk code seems to only call these operations on paths it knows are directories. Directory rename isn't going to create markers, nor do the S3A committers - therefore bulk data generated through any hive or spark commit operation will not have markers

Modification operations

`mkdir()`

We do walk up the tree looking for parents; we could maybe optimize it a bit.

S3A `innerMkdirs()` calls `getFileStatus()` to probe dest path for being a file or dir, then goes to reject/no-op.

The HEAD path + / in that code may add a 404 to the S3 load balancers, so subsequent probes for the path fail.
(Filed HADOOP-16879)

For now: don't worry about the 404 problem which doesn't seem to have created any known bug reports.

`rename(src, dest)`

1. rely on `innerGetFileStatus(path, needEmptyDirectoryFlag=true)` to probe correctly.
2. HADOOP-16493; skip the parent dir check on dest if dest.getParent == src.getParent and so skip a check

3. `finishRename()` to optionally delete markers up the tree

`rename(src, dest, opts)`

Let's get HADOOP-11452 done and switch to rename/3 in: hadoop MR committers, hive, distcp.. Places where the changes in renaming into an empty dir are handled correctly. Why the change: meaningful exceptions on all failed operations. Who wouldn't want that?

`delete()`

rely on `innerGetFileStatus(path, needEmptyDirectoryFlag=true)` to probe correctly.

We MUST NOT assume that a marker/ means that this is an empty dir, or that for a directory delete all which is needed is to delete that marker. We must list all children both to reject a non-recursive list and to start the bulk delete process.

`finishedWrite(key, length, eTag, versionId, BulkOperationState)`

This calls `deleteUnnecessaryFakeDirectories`(); deletion now needs to become optional.

`deleteUnnecessaryFakeDirectories(path)`

This is where we should check the path and decide based on our current marker retention policy, whether to issue a bulk request or not.

Or: add a `maybeDeleteUnnecessaryFakeDirectories(path)` to make clear its optional based on policy; retain `deleteUnnecessaryFakeDirectories()` as is.

`createFakeDirectoryIfNecessary(Path)`

Called at the end of delete & rename to reinstate a parent directory entry.

This only issues a LIST on the assumption that there probably isn't a fake marker and there's little cost to creating a fake one atop a fake one if somehow one was there. We should add the probe for the marker too now, as that is likely to become a common case.

Test Plan

All existing tests of filesystem semantics must succeed with both keep and delete options.

Issue: should we make the retention policy a new maven profile option? So contributed can test with either option, switch between them, and declare their policy in pull requests? It would seem a good idea as it allows some people to test in full backward compatibility mode and others to try the new option.

Some ITestS3AFileOperationCost test will fail as the change in ordering of probes will result in different metrics for list/metadata requests, and counters of deletion operations will decrease.

This is actually how we can observe that the changes are taking place.

Proposed: ITestS3AFileOperationCost to be parameterised on marker retention policy. When markers are expected to be deleted then we should expect the current Numbers of deletions, both the different probe outcomes.
What is critical is it in the backward compatibility mode: those deletion requests must take place.

By implementing the policy logic (conf value interpretation + path authoritativeness probes) into a standalone class, a unit test suite can validate its behaviour.

Actual behaviour can be done through integration tests, using file system metrics to count which operations have taken place, and getObjectMetadata calls on the store to probe for the presence of markers.

For compatibility tests, we may want to make a copy of the S3AFileSystem.s3GetFileStatus() implementations on both Hadoop 3.3 and its predecessors (earlier versions haven't had the work to avoid/reduce 404 markers being created). Having real copy and pastes those versions with let us do the checks in the hadoop-aws integration suites rather than anything more complicated.

Cherry picking support into older versions.

We should make older versions resilient to having a directory markers with children. They can still do all the existing create/delete/recreate logic -just not interpret a marker as an empty directory. The ongoing (March 23, 2020) implementation does all its work in s3GetFileStatus(), where it is deeply intertwined with the 404 logic of HADOOP-16490; we will effectively need to reimplement it in releases without that JIRA (or better: cherry pick that work too).

Update: Initial implementation findings 2020-04-01

Production code changes

These can be split into the getFileStatus calls (which can/should be backported) and then the directory marker retention code, which isn't required in older versions for backwards compatibility.

`getFileStatus()`
Changes on getFileStatus to avoid interpreting a directory marker as an empty directory. This will need to be backported/reimplemented on all older branches to

`S3AFileSystem` -new code
`S3ListRequest` (cleanup)
`StatusProbeEnum` tricky. ALL now means HEAD and LIST only, not marker dir. REALLY_ALL is for all three, and FILES_AND_DIRECTORIES what we should be using in ALL. (Note: not relevant for backport)

Changes for the new (configurable) marker retention/delete policy

`Constants` -new config
`RenameOperation` - skip deleting marker of destination parent (TODO)
`DirectoryMarkerRetention` - isolated/testable policy logic
Test setup: pass down `fs.s3a.directory.marker` option from `pom.xml` to set that property

Tests which break

`ITestS3AEncryptionSSEC` - getFileStatus calls on marker directories which would fail with encryption auth error on HEAD now succeed with LIST; assertions needed changing. This was also parameterized for s3guard on/off and markers keep/delete.
`ITestS3ARemoteFileChanged` - number- of HEAD requests to mock returning out of date information reduced
`ITestS3AFileOperationCost` -both on reduced ops on get/list and reduced #of delete operations.

Some S3AFileSystem code paths weren't finishing fast on root paths: fixed

The S3Guard nonauth past-TTL-probe-via-HEAD code didn't update S3Guard tables after the later HEAD if the file was unchanged. Which meant once the TTL expired, all getFileStatus/open, etc calls went to S3, even though S3Guard was up to date.
Now we update it so as to set the timestamp in the dynamoDB record to the current time, hence resetting the TTL.

We also had to improve S3A tombstone pruning to avoid the list of tombstones getting too big and overloading some of the dir LIST calls; whereas before this was only ever done during a "hadoop s3guard prune" call, now it is done incrementally after a set of tombstones have been collected.

we could/should do a further iteration there by making it something done async: the DDB metastore would have a thread to delete entries and we'd queue more for deletion; those already in the queue or recently deleted (retain in a cache) would not be double deleted. This will avoid reads being slowed by the need to update the store if many tombstones need to be cleaned up.

`S3AFileSystem.innerGetFileStatus()` is fairly convoluted code. The PR tries to clean this up by moving from a chain of operations which may/may not define some variables, followed by code which checks those variables being null into a more structured nested if/else tree. This should be clearer and easier to maintain.

Misc cleanups

`ITtlTimeProvider` -javadocs

Issue: backwards compatibility

We are not going to be able to do a trivial backport of the getFileStatus changes to any branch which does not have all of/some of the 404 handling, HADOOP-16490. That was where the `s3GetFileStatus(path)` was extended to take a set of probes (head, dir, list) and our FS operations tuned to only execute the ones we want.

Meaning: the `s3GetFileStatus()` code will be a rewrite for those older versions anyway.

Proposed

1. Before committing this patch to trunk, we also do the implementation for hadoop <= 3.2.x, which would become the patch for all previous versions.
2. From hadoop 3.2.0 we copy the existing `s3GetFileStatus()` call into a test method for interop testing "what breaks?"

3. And in trunk we replicate the new code we will add for backwards compatibility "have we fixed it?"

Also: log policy from outset, at least for now. We can remove that message at some point in the future, but for now I'd like
to see it in logs.

Issue: how much code restructuring to do?

Could we do this as part of "the ongoing refactoring?" It's really tempting. We could move `s3GetFileStatus()` into its own class with two callbacks into `S3AFilesystem.getObjectMetadata` and `ListObjects`, plus a factory method to construct the `ListRequest`. It's exactly the "layer below the FS API but above the S3 API calls" we want.

Pro

Isolate it and we have a smaller S3AFileSystem.java file to handle, and the ability to substitute a fake implementation of those callbacks, so, if we have understood the S3 API adequately (always doubtful), provide some unit tests of the logic. We can also reliably implement delayed visibility/404 caching failures.

Against that:

1. - we will make back porting an even bigger complication that is today. This is particularly true if we picked up use of `StoreContext` as its accessors to config/operations of the S3A FS instance,
2. - patch is big enough as it is.

If we were not worried about backporting the listing logic, I'd be tempted. But there's enough for people to review -and big patches don't get enough scrutiny just due to their size and effort it takes