Item 134
git_comments:

git_commits:

1. **summary:** [SPARK-5227] [SPARK-5679] Disable FileSystem cache in WholeTextFileRecordReaderSuite
   message: [SPARK-5227] [SPARK-5679] Disable FileSystem cache in WholeTextFileRecordReaderSuite This patch fixes two difficult-to-reproduce Jenkins test failures in InputOutputMetricsSuite (SPARK-5227 and SPARK-5679). The problem was that WholeTextFileRecordReaderSuite modifies the `fs.local.block.size` Hadoop configuration and this change was affecting subsequent test suites due to Hadoop's caching of FileSystem instances (see HADOOP-8490 for more details). The fix implemented here is to disable FileSystem caching in WholeTextFileRecordReaderSuite. Author: Josh Rosen <joshrosen@databricks.com> Closes #4599 from JoshRosen/inputoutputsuite-fix and squashes the following commits: 47dc447 [Josh Rosen] [SPARK-5227] [SPARK-5679] Disable FileSystem cache in WholeTextFileRecordReaderSuite

github_issues:

github_issues_comments:

github_pulls:

1. **title:** [SPARK-5227] [SPARK-5679] Disable FileSystem cache in WholeTextFileRecordReaderSuite
   body: This patch fixes two difficult-to-reproduce Jenkins test failures in InputOutputMetricsSuite (SPARK-5227 and SPARK-5679). The problem was that WholeTextFileRecordReaderSuite modifies the `fs.local.block.size` Hadoop configuration and this change was affecting subsequent test suites due to Hadoop's caching of FileSystem instances (see HADOOP-8490 for more details). The fix implemented here is to disable FileSystem caching in WholeTextFileRecordReaderSuite.

github_pulls_comments:

1. [Test build #27461 has started] (https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/27461/consoleFull) for PR 4599 at commit [`47dc447`] (https://github.com/apache/spark/commit/47dc4473cb817f5bf14a00f0c487c984aa2cc7c6). - This patch merges cleanly.
2. I SSH'ed into an AMPLab Jenkins box to reproduce the original failures and have confirmed that this patch fixes them. /cc @ksakellis
3. JIRA links, for the curious: - https://issues.apache.org/jira/browse/HADOOP-8490 - https://issues.apache.org/jira/browse/SPARK-5679 - https://issues.apache.org/jira/browse/SPARK-5227
4. [Test build #27461 has finished] (https://amplab.cs.berkeley.edu/jenkins/job/SparkPullRequestBuilder/27461/consoleFull) for PR 4599 at commit [`47dc447`] (https://github.com/apache/spark/commit/47dc4473cb817f5bf14a00f0c487c984aa2cc7c6). - This patch **passes all tests**. - This patch merges cleanly. - This patch adds no public classes.
5. Test PASSed. Refer to this link for build results (access rights to CI server needed): https://amplab.cs.berkeley.edu/jenkins//job/SparkPullRequestBuilder/27461/ Test PASSed.
6. /cc @pwendell @andrewor14
7. Ok, this LTGM
8. LGTM .

github_pulls_reviews:

1. So, do you think we should disable it across all tests? just in case there are other tests that also modify the hadoop configuration thinking that the config objects are local to them? It might bite someone else in the butt later if we don't globally do this. I don't think there is a global test class that every tests inherits, maybe we can add it in SparkSparkContext since a lot of the new tests written use that trait?

2. Good question. If we wanted to disable this in all tests, then I think the right place to do that would be in the Maven and SBT builds via system properties. I chose not to do that here because I wasn't sure whether doing so might mask bugs, since most users of Spark will run with FileSystem caching enabled (I think that disabling it across the board may harm performance, since it sounds like a lot of Hadoop code assumes that FileSystem.get is cheap, and, accordingly, calls it many times).
3. My opinion is that doing it for all tests would be a bit drastic. It's not that modifications to `Configuration` objects arbitrarily affect other `Configuration` objects. It's that this test specifically relies on some underlying properties set on the `FileSystem`, and the FS Cache allows these to leak to other FS instances.

jira_issues:

1. **summary:** Add Configuration to FileSystem cache key
   description: The {{FileSystem#get(URI, Configuration}} does not take the given {{Configuration}} into consideration before returning an existing fs instance from the cache with a possibly different conf.

jira_issues_comments:

1. Not honoring the given conf causes the obvious problem of not being able to tweak values. It's also causing problem for the NM. When an app is done, it should be able to call {{FileSystem.closeAllForUGI}} just like the JT does. Unfortunately that may pull the rug out from under another app for that user also running on the NM. It also means that multiple jobs for the same user are erroneously using the first job's conf. Both are probably latent issues in the JT but go unnoticed or are masked by retries. Ideally the {{hashCode}} or {{identityHashCode}} would be added to the cache key. A key/value equivalence test should not be performed because seemingly identical confs (ex. cloned from each other) would initially appear the same but may later change. One potential issue is cloned confs that really should be the same -- ex. yarn often creates a {{YarnConfiguration(conf)}}. This won't be a problem if the conversion is done once and stashed. If it's done on the fly multiple times, then it does present a problem. Arguably that would be a bug but it would be difficult to fix in a timely manner. So an alternative is to add a key to the conf (ex. {{fs.cache-id}}) that can be used in the fs cache key. This would allow partitioning of the cache, albeit imperfectly, that would account for cloned confs that should be treated the same. The onus is placed upon the caller to explicitly change the key when needed, but it would be more transparent for existing code. I'll wait for comments before preceding.
2. Making conf as part of the cache key solves number of problems, but may not be ideal by itself. We need to think about what are the parts of conf that make sense to be dynamic. Some might need to be allowed to change run-time, others need to stay the same once an input or output stream is created. Creating unique instances for all these variations may not be ideal. Probably FileSystem and the gut should only look at small static part of conf and allow each usage to tweak the behavior in a way that does not interfere with others that share objects/resources. I believe FileSystem cache was originally intended to work this way. The problem is, it currently doesn't. We need a better way to maintain this separation.
3. Good points, but I think the issue of sanely managing dynamic vs static values is a separate issue. On that tangent, {{FileSystem}} probably isn't a good place to make that decision since the actual underlying clients of the filesystems are the ones that need to make that determination. It's a much harder problem than this one. In this jira, I'm only proposing that a user can take specific action to ensure a unique yet cached fs instance can be created and later closed.
4. bq. In this jira, I'm only proposing that a user can take specific action to ensure a unique yet cached fs instance can be created and later closed. Couldn't the user just use {{FileSystem.newInstance}}, so that it doesn't use the cache?
5. It sounds like that NM has a specific need on caching. It may make sense to disable FileSystem cache (using fs.hdfs.impl.disable.cache) for NM and then it implements its own cache. Adding conf as a key to current FileSystem cache may break many existing applications.
6. The cache partitioning would be extremely useful when you _do want_ the fs instance cached, but don't want it to collide with other cached instances. Ex. you want to be able to close all the filesystems that have been acquired after an operation. It's not a one-off thing in the NM. It would also be nice for the FsShell to get a partition in the cache to avoid closing a caller's filesystems. One problem is when code does a {{FileSystem.get(...)}} in many places and expects to get the same fs instance. Using {{newInstance}}, which I originally thought of, does not address that case w/o modifying all the code (which isn't always feasible) to pass around and use that fs instance. Code that does {{Path#getFileSystem(...)}} won't necessarily know up front which unique instances to obtain and then

converting the code to pass around and use the list of unique instances would be extremely painful. Basically the user is forced to maintain their own cache of filesystems... bq. Adding conf as a key to current FileSystem cache may break many existing applications. How so? The cache key is invisible/private?

7. I think that it would be nice to actually fix the cache if at all possible. I have heard that most multi-tenet long running systems that use HDFS have to disable the cache and then write their own, i.e. Oozie and Hive. The JobTracker is the only one I know of that does not do this. Most client code I have seen really relies on the fact that getting a "new" FileSystem is cheap, so completely removing the cache is not a feasible option. Look at the MR/Yarn code. Just uploading aggregated log files to HDFS creates about 5 FileSystems if the cache is disabled. If our users have to disable and work around a "feature" that we cannot turn off we should take that as a bad sign and try to provide a better solution for them. Now if the "fix" makes the performance horrible, or there are other problems we may need to rethink things, but I am +1 on trying to fix the cache.
8. Specific issue that prompted the bug was fixed in the NM long ago.