Item 33
git_comments:

1. Bulk Load methods
2. * * Close all the readers We don't need to worry about subsequent requests because the HRegion * holds a write lock that will prevent any more reads or writes. * @return the {@link StoreFile StoreFiles} that were previously being used. * @throws IOException on failure
3. * * Adds a value to the memstore * @param kv * @return memstore size delta
4. * * This throws a WrongRegionException if the HFile does not fit in this region, or an * InvalidHFileException if the HFile is not valid.
5. * * Used for tests. * @return cache configuration for this Store.
6. * * Return a scanner for both the memstore and the HStore files. Assumes we are not in a * compaction. * @param scan Scan to apply when scanning the stores * @param targetCols columns to scan * @return a scanner over the current key values * @throws IOException on failure
7. * * See if there's too much store files in this store * @return true if number of store files is greater than the number defined in minFilesToCompact
8. * * @return <tt>true</tt> if the store has any underlying reference files to older HFiles
9. * * Adds or replaces the specified KeyValues. * <p> * For each KeyValue specified, if a cell with the same row, family, and qualifier exists in * MemStore, it will be replaced. Otherwise, it will just be inserted to MemStore. * <p> * This operation is atomic on each KeyValue (row/family/qualifier) but not necessarily atomic * across all of them. * @param kvs * @return memstore size delta * @throws IOException
10. * * @return true if we should run a major compaction.
11. * * Updates the value for the given row/family/qualifier. This function will always be seen as * atomic by other readers because it only puts a single KV to memstore. Thus no read/write * control necessary. * @param row row to update * @param f family to update * @param qualifier qualifier to update * @param newValue the new value to set into memstore * @return memstore size delta * @throws IOException
12. * * Interface for objects that hold a column family in a Region. Its a memstore and a set of zero or * more StoreFiles, which stretch backwards over time.
13. Split oriented methods
14. * * @param priority priority to check against. When priority is {@link HStore#PRIORITY_USER}, * {@link HStore#PRIORITY_USER} is returned. * @return The priority that this store has in the compaction queue.
15. General Accessors
16. * * Returns the total byte size of all Bloom filter bit arrays. For compound Bloom filters even the * Bloom blocks currently not loaded into the block cache are counted. * @return the total size of all Bloom filters in the store
17. * * @return the parent region hosting this store
18. **comment:** General accessors into the state of the store TODO abstract some of this out into a metrics class
    label: code-design
19. Compaction oriented methods
20. * * Find the key that matches <i>row</i> exactly, or the one that immediately precedes it. WARNING: * Only use this method on a table where writes occur with strictly increasing timestamps. This * method assumes this pattern of writes in order to make it reasonably performant. Also our * search is dependent on the axiom that deletes are for cells that are in the container that * follows whether a memstore snapshot or a storefile, not for the current container: i.e. we'll * see deletes before we come across cells we are to delete. Presumption is that the * memstore#kvset is processed before memstore#snapshot and so on. * @param row The row key of the targeted row. * @return Found keyvalue or null if none found. * @throws IOException
21. * * Licensed to the Apache Software Foundation (ASF) under one * or more contributor license agreements. See the NOTICE file * distributed with this work for additional information * regarding copyright ownership. The ASF licenses this file * to you under the Apache License, Version 2.0 (the * "License"); you may not use this file except in compliance * with the License. You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License.
22. * * Returns the total size of all index blocks in the data block indexes, including the root level, * intermediate levels, and the leaf level for multi-level indexes, or just the root level for * single-level indexes. * @return the total size of block indexes in the store

23. * * getter for CompactionProgress object * @return CompactionProgress object; can be null
24. * * This method should only be called from HRegion. It is assumed that the ranges of values in the * HFile fit within the stores assigned region. (assertBulkLoadHFileOk checks this)
25. * * Removes a kv from the memstore. The KeyValue is removed only if its key & memstoreTS match the * key & memstoreTS value of the kv parameter. * @param kv
26. Test-helper methods
27. * * Compact the most recent N files. Used in testing. * @param N number of files to compact. Must be less than or equal to current number of files. * @throws IOException on failure

git_commits:

1. **summary:** HBASE-6569 Extract HStore interface from Store (Jesse Yates)
   message: HBASE-6569 Extract HStore interface from Store (Jesse Yates) git-svn-id: https://svn.apache.org/repos/asf/hbase/trunk@1373153 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Extract HStore interface from Store
   description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
   label: code-design
2. **summary:** Extract HStore interface from Store
   description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
   label: code-design
3. **summary:** Extract HStore interface from Store
   description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
4. **summary:** Extract HStore interface from Store
   description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
5. **summary:** Extract HStore interface from Store
   description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
6. **summary:** Extract HStore interface from Store

description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).

7. **summary:** Extract HStore interface from Store
   description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
   label: code-design
8. **summary:** Extract HStore interface from Store
   description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
   label: code-design
9. **summary:** Extract HStore interface from Store
   description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
10. **summary:** Extract HStore interface from Store
    description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
11. **summary:** Extract HStore interface from Store
    description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
12. **summary:** Extract HStore interface from Store
    description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
    label: code-design
13. **summary:** Extract HStore interface from Store
    description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
14. **summary:** Extract HStore interface from Store
    description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
15. **summary:** Extract HStore interface from Store
    description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the

amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).

16. **summary:** Extract HStore interface from Store
    description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).
    label: code-design
17. **summary:** Extract HStore interface from Store
    description: Currently Store.java is a top-level class. However, we need to to use the same interface for snapshots (HBASE-6055), and potentially other uses-cases as well, but subclassing is 'dirty' given the amount of state that a Store currently builds up on instantiation. For snapshots, we just need to modify the Store actions slightly, making a composition with interface inheritance design very appealing (meaning we need an interface for Store).

jira_issues_comments:

1. **body:** Attaching patch, posting to RB momentarily. This touches a lot of files, but is 99% just a naming change.
   label: code-design
2. **body:** +1 after addressing Ted's comments on RB about inconsistent use of 'public' and interface annotations.
   label: code-design
3. Patch from review board. I am running test suite as well. Will integrate if there is no objection.
4. Test suite has completed. TestFromClientSideWithCoprocessor#testNonCachedGetRegionLocation failed but passed when run individually. Integrated to trunk. Thanks for the patch, Jesse. Thanks for the review, Andy.
5. Integrated in HBase-TRUNK #3216 (See [https://builds.apache.org/job/HBase-TRUNK/3216/]) HBASE-6569 Extract HStore interface from Store (Jesse Yates) (Revision 1373153) Result = FAILURE tedyu : Files : * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/CompactSplitThread.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/CompactionRequestor.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/ConstantSizeRegionSplitPolicy.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegionServer.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HStore.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/IncreasingToUpperBoundRegionSplitPolicy.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/MemStore.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/MemStoreFlusher.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/RegionSplitPolicy.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/Store.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/backup/example/TestZooKeeperTableArchiveClient.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/client/TestFromClientSide.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/CompactionTool.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestAtomicOperation.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestCompaction.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestHRegion.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestRegionServerMetrics.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/wal/TestWALReplay.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/util/HFileArchiveTestingUtil.java
6. Integrated in HBase-TRUNK-on-Hadoop-2.0.0 #130 (See [https://builds.apache.org/job/HBase-TRUNK-on-Hadoop-2.0.0/130/]) HBASE-6569 Extract HStore interface from Store (Jesse Yates) (Revision 1373153) Result = FAILURE tedyu : Files : * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/CompactSplitThread.java *

/hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/CompactionRequestor.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/ConstantSizeRegionSplitPolicy.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegionServer.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HStore.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/IncreasingToUpperBoundRegionSplitPolicy.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/MemStore.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/MemStoreFlusher.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/RegionSplitPolicy.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/Store.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/backup/example/TestZooKeeperTableArchiveClient.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/client/TestFromClientSide.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/CompactionTool.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestAtomicOperation.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestCompaction.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestHRegion.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestRegionServerMetrics.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/wal/TestWALReplay.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/util/HFileArchiveTestingUtil.java

7. -1 overall. Here are the results of testing the latest attachment http://issues.apache.org/jira/secure/attachment/12540950/6569-v3.patch against trunk revision . +1 @author. The patch does not contain any @author tags. +1 tests included. The patch appears to include 27 new or modified tests. -1 patch. The patch command could not apply the patch. Console output: https://builds.apache.org/job/PreCommit-HBASE-Build/2571//console This message is automatically generated.
8. The Interface should have been called ColumnFamily? Mind if I change it Jesse?
9. **body:** That's closer, but its still kind of a leaky abstraction if you call it ColumnFamily. Maybe ColumnFamilyManager (something along those lines)?
   label: code-design
10. **body:** @Jesse OK. I'm fine w/ keeping logical and implementation distinct -- Store vs ColumnFamily. I'm reacting mostly to the name given the Interface, HStore. If the Interface is to have a letter prefix, can it at least be I for Interface? As ugly as this is, it is better than 'H'. Or StoreInterface? (Minor point is that Store used to be called HStore. I worked hard to rename it a while ago. I was trying to undo our prefixing everything w/ the redundant 'H'. Thats not important)
    label: code-design
11. +1 on StoreInterface
12. Agree with Ted (and stack). I'm generally against the smurf naming (smurf-berries, smurf-region, smurf-master, etc), but was just trying to keep with convention as it appeared.
13. Shouldn't it then be "Store" and "StoreImpl"? (Note that this falls squarely in the "I couldn't care less" category for me, just saying that's what we used in some other places, maybe we need to generally agree on a convention for HBase).
14. **body:** I will make a patch that renames the Interface 'Store' and the implementation 'HStore'. HStore goes with HRegion, etc. Yes, its smurf naming but I think this better than my IStore/StoreInterface and StoreImpl....
    label: code-design
15. @stack can we close this issue?
16. The final rename was done in HBASE-6599. Closing this one.
17. Marking closed.