Item 39
git_comments:

1. * * * 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.
2. * * Create an HFile with the given number of rows between a given * start key and end key.
3. * * Utility class for HFile-related testing.
4. * * Convenience method, the equivalent of checking if result is * FLUSHED_NO_COMPACTION_NEEDED or FLUSHED_NO_COMPACTION_NEEDED. * @return true if the memstores were flushed, else false.
5. Be careful adding more to this enum, look at the below methods to make sure
6. We need to assign a sequential ID that's in between two memstores in order to preserve the guarantee that all the edits lower than the highest sequential ID from all the HFiles are flushed on disk. See HBASE-10958.
7. Special case where a flush didn't run because there's nothing in the memstores. Used when bulk loading to know when we can still load even if a flush didn't happen.
8. * * Convenience constructor to use when the flush is successful, the failure message is set to * null. * @param result Expecting FLUSHED_NO_COMPACTION_NEEDED or FLUSHED_COMPACTION_NEEDED. * @param flushSequenceId Generated sequence id that comes right after the edits in the * memstores.
9. * * Convenience constructor to use when we cannot flush. * @param result Expecting CANNOT_FLUSH_MEMSTORE_EMPTY or CANNOT_FLUSH. * @param failureReason Reason why we couldn't flush.
10. * * Constructor with all the parameters. * @param result Any of the Result. * @param flushSequenceId Generated sequence id if the memstores were flushed else -1. * @param failureReason Reason why we couldn't flush, or null.
11. * * Objects from this class are created when flushing to describe all the different states that * that method ends up in. The Result enum describes those states. The sequence id should only * be specified if the flush was successful, and the failure message should only be speficied * if it didn't flush.
12. * * Convenience method, the equivalent of checking if result is FLUSHED_COMPACTION_NEEDED. * @return True if the flush requested a compaction, else false (doesn't even mean it flushed).
13. empty memstore, flush doesn't run
14. Two flushes after the threshold, compactions are needed
15. Flush enough files to get up to the threshold, doesn't need compactions
16. * * Test that we get the expected flush results back * @throws IOException
17. major compact to turn all the bulk loaded files into one normal file
18. Add an edit so something in the WAL
19. * * HRegion test case that is made of a major compacted HFile (created with three bulk loaded * files) and an edit in the memstore. * This is for HBASE-10958 "[dataloss] Bulk loading with seqids can prevent some log entries * from being replayed" * @throws IOException * @throws IllegalAccessException * @throws NoSuchFieldException * @throws IllegalArgumentException * @throws SecurityException
20. Now 'crash' the region by stealing its wal
21. I can't close wal1. Its been appropriated when we split.

git_commits:

1. **summary:** HBASE-10958 [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
   message: HBASE-10958 [dataloss] Bulk loading with seqids can prevent some log entries from being replayed git-svn-id: https://svn.apache.org/repos/asf/hbase/branches/0.98@1590145 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
   description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
   label: code-design
2. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed

description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.

3. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
   description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
   label: code-design
4. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
   description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
   label: code-design
5. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
   description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
6. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
   description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
7. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
   description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a

second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
label: code-design

8. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
label: test
9. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
label: code-design
10. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
label: test
11. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
12. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
label: code-design
13. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is

higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
label: code-design

14. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
15. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
16. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
17. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
18. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.

19. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
    description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
20. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
    description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
21. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
    description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
22. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
    description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
23. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
    description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
24. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
    description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the

region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.

25. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
26. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
27. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
28. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
29. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
30. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets

seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.

31. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
32. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
label: code-design
33. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
34. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
35. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
36. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than

unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
label: code-design

37. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
38. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
label: code-design
39. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
40. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
41. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
42. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed

description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.

43. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
44. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
45. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
46. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.
47. **summary:** [dataloss] Bulk loading with seqids can prevent some log entries from being replayed
description: We found an issue with bulk loads causing data loss when assigning sequence ids (HBASE-6630) that is triggered when replaying recovered edits. We're nicknaming this issue *Blindspot*. The problem is that the sequence id given to a bulk loaded file is higher than those of the edits in the region's memstore. When replaying recovered edits, the rule to skip some of them is that they have to be _lower than the highest sequence id_. In other words, the edits that have a sequence id lower than the highest one in the store files *should* have also been flushed. This is not the case with bulk loaded files since we now have an HFile with a sequence id higher than unflushed edits. The log recovery code takes this into account by simply skipping the bulk loaded files, but this "bulk loaded status" is *lost* on compaction. The edits in the logs that have a sequence id lower than the bulk loaded file that got compacted are put in a blind spot and are skipped during replay. Here's the easiest way to recreate this issue: - Create an empty table - Put one row in it (let's say it gets seqid 1) - Bulk load one file (it gets seqid 2). I used ImporTsv and set hbase.mapreduce.bulkload.assign.sequenceNumbers. - Bulk load a second file the same way (it gets seqid 3). - Major compact the table (the new file has seqid 3 and isn't considered bulk loaded). - Kill the

region server that holds the table's region. - Scan the table once the region is made available again. The first row, at seqid 1, will be missing since the HFile with seqid 3 makes us believe that everything that came before it was flushed.

jira_issues_comments:

1. **body:** One workaround we found is to completely disable compactions, then when you need to run them you have to force flush the regions that have bulk loaded file first and ensure that bulk loads aren't coming in at the same time. Workloads that are strictly doing incremental bulk loads aren't affected, you need a mix of bulk loaded files and normal Puts. A hacky solution could be to force flush when bulk loading with seqids and grab the next sequence id that comes after the memstore flush to go to the bulk loaded file. This means that bulk loading needs to initiate a flush, get the sequence id under the region write lock, then do the bulk load. We don't need to wait for the flush to happen... unless the possibility for the bulk loaded file to be compacted before the flush is done is high enough.
   label: code-design
2. Seems fine to force a flush before bulk loading.
3. **body:** Here's a quick hack I put together to show one solution. In this case I inline a flush with the {{bulkLoadHFiles}} call and I modified the {{internalFlushcache}} code to be able to get more state back and also return a sequential ID that ends up being in between two memstores. I tested that it works following the steps listed in this jira's description. I don't really like having to wait for the flush to happen since it could take a lot of time to finish making bulk loading way slower. On the other hand, it's safer than just requesting a flush asynchronously and then grabbing a new sequence ID from HLog. Maybe it would still be fine since you need to also have a compaction to trigger the bug...
   label: code-design
4. **body:** Here's a less intrusive hack that shows what it looks like to request an asynchronous flush from {{bulkLoadHFiles}}. I also tested that it works like the previous patch. The benefits are that the bulk loader doesn't have to wait and it's a lot less code, but the blindspot is still there and it widens as the region server gets more load; for example, if a lot of flushing is happening, which is likely with this patch, then the flush requests are queued and a compacting can kick in and at that moment if the region server dies then data is lost.
   label: code-design
5. I'll throw my 2 canadien cents and say that I prefer the "removing-the-second-lane" solution (the one where we do a synchronous flush prior to bulk load). My rationale is that the other approach still leaves us with a blindspot, even if greatly reduced and we would effectively still have the bug (potentially).
6. I'm w/ [~alexandre.normand]. FlushState should be FlushResult? Does it have to be public? Should below be HBaseIOE (you know who is watching) or some specialization on HBaseIOE, FlushFailedIOE(FlushState)? throw new IOException(What is happening here: + seqId = fs.flushSequenceId; This is the 'next' seqid after the flush or the seqid that was written into the hfile that was flushed? Do we need to up the seqid if the flush one is being used for a bulk load so the next edit in memstore has a different number? Or that is done already elsewhere? Good stuff JD. A unit test would be too hard to conjure? Maybe describe then instead how you replicate.
7. **body:** bq. FlushState should be FlushResult? Does it have to be public? Some external classes to that package call {{flushcache()}} directly. Also, that method is public so whatever it returns needs to be as visible. bq. Should below be HBaseIOE (you know who is watching) or some specialization on HBaseIOE, FlushFailedIOE(FlushState)? It's inline with the rest of that method, which the bulk loading client seems to process correctly. Not that it shouldn't be considered, but maybe in a different jira? {quote} This is the 'next' seqid after the flush or the seqid that was written into the hfile that was flushed? Do we need to up the seqid if the flush one is being used for a bulk load so the next edit in memstore has a different number? Or that is done already elsewhere? {quote} Yeah that part is hard to follow, needs at least some documentation (hey it is a hack!). So it's set here: + return new FlushState(flushSeqId, compactionRequested); That {{flushSeqId}} comes from here: {code} Long startSeqId = wal.startCacheFlush(this.getRegionInfo().getEncodedNameAsBytes()); if (startSeqId == null) { status.setStatus("Flush will not be started for [" + this.getRegionInfo().getEncodedName() + "] - WAL is going away"); return false; } flushSeqId = startSeqId.longValue(); {code} So it's a sequence id that's the same one as the one used to signal the flush. I thought about creating a new one just after, but I'm not sure if it's necessary since that {{startSeqId}} will be after all the MemStore edits. bq. A unit test would be too hard to conjure? Maybe describe then instead how you replicate. Doesn't seem too hard to write... at least the recreating what I'm doing manually shouldn't take too long to write, but testing the corner cases seems much harder. I'm still wondering if there's a more elegant solution.
   label: code-design
8. **body:** Patch that addresses most of Stack's comments and based on trunk. Still no unit test, spent too much time today trying to understand why {{TestHRegion}} fails so much (see HBASE-10312).
   label: test
9. **body:** {color:red}-1 overall{color}. Here are the results of testing the latest attachment http://issues.apache.org/jira/secure/attachment/12639893/HBASE-10958.patch against trunk revision . ATTACHMENT ID: 12639893 {color:green}+1 @author{color}. The patch does not contain any @author tags. {color:green}+1 tests included{color}. The patch appears to include 6 new or modified tests. {color:green}+1 javadoc{color}. The javadoc tool did not generate any warning messages. {color:green}+1 javac{color}. The applied patch does not increase the total number of javac compiler warnings. {color:red}-1 findbugs{color}. The patch appears to introduce 2 new Findbugs (version 1.3.9) warnings. {color:green}+1 release audit{color}. The applied patch does not increase the total number of release audit warnings. {color:green}+1 lineLengths{color}. The patch does not introduce lines longer than 100 {color:green}+1 site{color}. The mvn site goal succeeds with this patch. {color:red}-1 core tests{color}. The patch failed these unit tests: org.apache.hadoop.hbase.security.access.TestAccessController Test results: https://builds.apache.org/job/PreCommit-HBASE-Build/9267//testReport/ Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9267//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-protocol.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9267//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-thrift.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9267//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-client.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9267//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-hadoop2-compat.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9267//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-examples.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9267//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-prefix-tree.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9267//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-common.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9267//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-server.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9267//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-hadoop-compat.html Console output: https://builds.apache.org/job/PreCommit-HBASE-Build/9267//console This message is automatically generated.
   label: code-design

10. **body:** New patch with a unit test inside {{TestWALReplay}}. I've done a bit of refactoring and I'm tempted to go a step further and somehow extract the common bits from testRegionMadeOfBulkLoadedFilesOnly and testCompactedBulkLoadedFiles but it seems a bit messy.
    label: test
11. Oh and I found that testRegionMadeOfBulkLoadedFilesOnly wasn't really testing WAL replay with a bulk loaded file, the KV was being added at LATEST_TIMESTAMP so it wasn't visible being so far in the future. I made it so we check that we got all the rows back.
12. **body:** Patch looks great [~jdcryans]. Minor nit is that you do + if (fs.flushSucceeded()) { + seqId = fs.flushSequenceId; + } else if (fs.result == FlushResult.Result.CANNOT_FLUSH_MEMSTORE_EMPTY) { flushSucceeded method (should it be isFlushSucceeded) and then you go get the result by accessing the data member directly. Minor inconsistency. +1 on commit if hadoopqa ok. Can address above if you want on commit.
    label: code-design
13. **body:** {color:red}-1 overall{color}. Here are the results of testing the latest attachment http://issues.apache.org/jira/secure/attachment/12640324/HBASE-10958-v2.patch against trunk revision . ATTACHMENT ID: 12640324 {color:green}+1 @author{color}. The patch does not contain any @author tags. {color:green}+1 tests included{color}. The patch appears to include 9 new or modified tests. {color:green}+1 javadoc{color}. The javadoc tool did not generate any warning messages. {color:green}+1 javac{color}. The applied patch does not increase the total number of javac compiler warnings. {color:green}+1 findbugs{color}. The patch does not introduce any new Findbugs (version 1.3.9) warnings. {color:green}+1 release audit{color}. The applied patch does not increase the total number of release audit warnings. {color:green}+1 lineLengths{color}. The patch does not introduce lines longer than 100 {color:green}+1 site{color}. The mvn site goal succeeds with this patch. {color:red}-1 core tests{color}. The patch failed these unit tests: org.apache.hadoop.hbase.security.access.TestAccessController Test results: https://builds.apache.org/job/PreCommit-HBASE-Build/9296//testReport/ Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9296//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-protocol.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9296//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-thrift.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9296//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-client.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9296//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-hadoop2-compat.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9296//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-examples.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9296//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-prefix-tree.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9296//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-common.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9296//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-server.html Findbugs warnings: https://builds.apache.org/job/PreCommit-HBASE-Build/9296//artifact/trunk/patchprocess/newPatchFindbugsWarningshbase-hadoop-compat.html Console output: https://builds.apache.org/job/PreCommit-HBASE-Build/9296//console This message is automatically generated.
    label: code-design
14. That's a real test failure, the user that bulk loads doesn't have the permission to flush. Looking.
15. Oh; that's an unforeseen problem. Does it make sense for a user to be able to bulkload but not to flush? (I suppose it does as bulk loading is not in principle different from inserting data via Put/Delete).
16. Yeah, and bulk loading itself is kind of like a flush since you end up with an HFile.
17. Right now bulk loading is WRITE and flush is ADMIN. Problematic!
18. Solutions on top of my head: - Do doAs inside the method. The rationale being that it's the region server that's trying to do that here, not the user. I'm not sure if this is doable, and [~mbertozzi] is laughing at me. - Matteo thinks that we should just make bulk load an ADMIN method. I agree but I'm not fond of breaking secure setups that use bulk loads. [~apurtell]?
19. Wait. Are you saying the region server itself cannot issue a flush as part of the bulkLoadHFiles RPC? The region server is flushing all the time on its own behalf (when the memstore is full, etc).
20. bq. Wait. Are you saying the region server itself cannot issue a flush as part of the bulkLoadHFiles RPC? Exact. That's why the User.runAs (and then run as the region server itself) solution seems to make sense to me. I read some more code, and it seems bulk load actually requires CREATE even though the call itself requires WRITE. From TestAccessController: {code} // User performing bulk loads must have privilege to read table metadata // (ADMIN or CREATE) verifyAllowed(bulkLoadAction, SUPERUSER, USER_ADMIN, USER_OWNER, USER_CREATE); verifyDenied(bulkLoadAction, USER_RW, USER_NONE, USER_RO); {code} So another options is to set flush and compact as CREATE actions (in line with create table, alter, disable, enable, delete).
21. I guess I did not realize that the authorization extends to any action issued from an RPC, rather than just authorizing the RPC call itself.
22. We expect READ and WRITE perms granted in a fine grained way to constraint who can do individual ops that only collectively add up to cluster impacting events like compactions, splits, and flushes. For actions that can have a global cluster impact, we'd like ADMIN to be granted sparingly to admins or delegates. IIRC enable and disable are ADMIN actions also, since disabling or enabling a 10000 region table has consequences. CREATE is kind of a middle ground for schema reads and updates, but in terms of schema update that's splitting hairs I suppose since a schema update of said large table would also have consequences of the same scale. Bulk load is a special snowflake because it's a series of puts (so, WRITE) yet obviously more than that as mentioned, we need to flush, and moving files in place will probably kick off compaction. Making bulk load an ADMIN action, or CREATE, makes sense to me also.
23. And the bit about needing CREATE or ADMIN to read schema metadata, this is to protect potentially sensitive information in the metadata that an ordinary user granted only READ or READ+WRITE access to the table has no need to see, that was HBASE-8692
24. Very nice explanation and insights on these permissions! Looking previously at this page: https://hbase.apache.org/book/hbase.accesscontrol.configuration.html we only have pretty vague info over there. Also, the 'write' permission for 'flush' and 'compact' in 'Table 8.1'. Are they even correct?
25. bq. we'd like ADMIN to be granted sparingly to admins or delegates +1 bq. IIRC enable and disable are ADMIN actions also, since disabling or enabling a 10000 region table has consequences. This is what I see in the code in trunk: {code} requirePermission("preBulkLoadHFile", ...getTableDesc().getTableName(), el.getFirst(), null, Permission.Action.WRITE); requirePermission("enableTable", tableName, null, null, Action.ADMIN, Action.CREATE); requirePermission("disableTable", tableName, null, null, Action.ADMIN, Action.CREATE); requirePermission("compact", getTableName(e.getEnvironment()), null, null, Action.ADMIN); requirePermission("flush", getTableName(e.getEnvironment()), null, null, Action.ADMIN); {code} IMO flush should have lower or same perms as disableTable. So here's a list of changes I believe are needed: - preBulkLoadHFile goes from WRITE to CREATE (seems more in line with what's really needed to bulk load given the code I posted yesterday) - compact/flush go from ADMIN to ADMIN or CREATE This should not have an impact on the current users. If we can agree on the changes, I'll open a new jira that's going to be blocking this one.

26. Maybe "CREATE" no longer expresses what it now implies...? I can see that folks would not want to grant users CREATE (or ADMIN) so that they cannot create/drop/enable/disable tables, but still do allow them to load data via bulk load. That would now no longer possible. Also it would seem more sensible to me that if a user bulk loads some data and then for *technical* reasons the region server decides to flush there should be no additional right needed; just a user does not need permission to flush only because a Put happens to cause a flush. Have we dismissed this option?
27. bq. I can see that folks would not want to grant users CREATE (or ADMIN) so that they cannot create/drop/enable/disable tables, but still do allow them to load data via bulk load. That would now no longer possible. Bulk load already needs CREATE as shown above in TestAccessController. bq. Have we dismissed this option? I don't think so, but no one has been pushing for it. It creates a precedent, nowhere else in the code do we use User.runAs to override the current user that came in via a RPC (I'm not even sure if it works, but that's because I don't know that code very well).
28. Missed the test comment. In that case let's do what you suggest. (Since this in an existing issue I might sill want release 0.94.19 before we fix it depending on whether this needs more discussion)
29. Just did a quick test. The requirement on 'CREATE' for bulk load seems to come from here. Is this even intended? {code} Exception in thread "main" org.apache.hadoop.hbase.security.AccessDeniedException: org.apache.hadoop.hbase.security.AccessDeniedException: Insufficient permissions (user=user1@IBM.COM, scope=TestTable, family=, action=CREATE) at org.apache.hadoop.hbase.security.access.AccessController.requirePermission(AccessController.java:356) at org.apache.hadoop.hbase.security.access.AccessController.preGetTableDescriptors(AccessController.java:1513) at org.apache.hadoop.hbase.master.MasterCoprocessorHost.preGetTableDescriptors(MasterCoprocessorHost.java:1260) at org.apache.hadoop.hbase.master.HMaster.getTableDescriptors(HMaster.java:2569) at org.apache.hadoop.hbase.protobuf.generated.MasterProtos$MasterService$2.callBlockingMethod(MasterProtos.java:40438) at org.apache.hadoop.hbase.ipc.RpcServer.call(RpcServer.java:2150) ... at org.apache.hadoop.hbase.protobuf.ProtobufUtil.getRemoteException(ProtobufUtil.java:235) at org.apache.hadoop.hbase.client.HConnectionManager$HConnectionImplementation.getHTableDescriptor(HConnectionManager.java:2632) at org.apache.hadoop.hbase.client.HTable.getTableDescriptor(HTable.java:548) at org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles.doBulkLoad(LoadIncrementalHFiles.java:233) at org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles.run(LoadIncrementalHFiles.java:820) {code}
30. bq. The requirement on 'CREATE' for bulk load seems to come from here. Is this even intended? Thanks for doing this. There's also another place where this is called, splitStoreFile(), in order to get an HCD. I don't think it's necessary to call it twice, but it seems necessary to call it at least once else you can't: - verify that the families exist - get the schema for each family so that we create the HFiles with the correct configurations when splitting
31. Forgot to reply to [~stack]'s comment: bq. flushSucceeded method (should it be isFlushSucceeded) and then you go get the result by accessing the data member directly. Minor inconsistency. flushSucceeded() isn't just looking up a field though, it's checking two things.
32. **body:** bq. IMO flush should have lower or same perms as disableTable. That seems fine. bq. Maybe "CREATE" no longer expresses what it now implies...? At the least we have an imperfect idea of when a user should be able to create tables and administer them, just "not administer them too much"
    label: code-design
33. bq. The requirement on 'CREATE' for bulk load seems to come from [getTableDescriptors]. Is this even intended? Yes. CREATE is overloaded to mean "RESTRICTED ADMIN", with ADMIN-ish privilege required because table schema is considered potentially sensitive. On another issue Francis Liu and I discussed the notion of creating a new permission 'SCHEMA' which would grant permission to read schema metadata. Now as then it seems maybe not quite needed (yet). CREATE and ADMIN would have such SCHEMA permission implicitly, so how useful would SCHEMA be, and there would still need a grant beyond WRITE for bulk loading.
34. Attaching 2 patches. One is a backport for 0.94. While doing the backport I saw that a TestSnapshotFromMaster was failing and Matteo was able to see that it was an error in my patch, flushing always returned that it needed compaction. I need to rerun all the tests now but it was the only one that failed (haven't tried with security either). I also added a test in TestHRegion for that. The second patch is for trunk, in which I ported the same test to TestHRegion. Interestingly, it didn't work. I found that in 0.94 we compact if num_files > compactionThreshold, but in trunk it's >=, so it seems that we compact more often now. This patch also has the fixes from [~stack]'s comments.
35. {color:red}-1 overall{color}. Here are the results of testing the latest attachment http://issues.apache.org/jira/secure/attachment/12640751/HBASE-10958-0.94.patch against trunk revision . ATTACHMENT ID: 12640751 {color:green}+1 @author{color}. The patch does not contain any @author tags. {color:green}+1 tests included{color}. The patch appears to include 15 new or modified tests. {color:red}-1 patch{color}. The patch command could not apply the patch. Console output: https://builds.apache.org/job/PreCommit-HBASE-Build/9328//console This message is automatically generated.
36. **body:** FYI rather than do this: + String method = "testFlushResult"; You can do this: method = name.getMethodName(); ... because in TestHRegion it does this: @Rule public TestName name = new TestName(); See here http://stackoverflow.com/questions/473401/get-name-of-currently-executing-test-in-junit-4 On the sometime an accessor, sometime not... not going to argue. nit. Patch LGTM (where G==Great)
    label: code-design
37. bq. method = name.getMethodName(); Will fix (looks like I copied that from one of the few methods in that class that doesn't do it).
38. **body:** bq. Will fix (looks like I copied that from one of the few methods in that class that doesn't do it). it is a nit. fix on commit?
    label: code-design
39. Nice. +1
40. Moving to 0.94.20.
41. Committed to 0.96 and up. Like for HBASE-11008, I'm waiting to commit to 0.94 or I can open a backport jira.
42. FAILURE: Integrated in HBase-TRUNK #5118 (See [https://builds.apache.org/job/HBase-TRUNK/5118/]) HBASE-10958 [dataloss] Bulk loading with seqids can prevent some log entries from being replayed (jdcryans: rev 1590144) * /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/HStore.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/RSRpcServices.java * /hbase/trunk/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/StoreFile.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/mapreduce/TestLoadIncrementalHFiles.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/wal/TestWALReplay.java * /hbase/trunk/hbase-server/src/test/java/org/apache/hadoop/hbase/util/HFileTestUtil.java
43. SUCCESS: Integrated in HBase-0.98 #296 (See [https://builds.apache.org/job/HBase-0.98/296/]) HBASE-10958 [dataloss] Bulk loading with seqids can prevent some log entries from being replayed (jdcryans: rev 1590145) * /hbase/branches/0.98/hbase-

server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/branches/0.98/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegionServer.java * /hbase/branches/0.98/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HStore.java * /hbase/branches/0.98/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/MemStoreFlusher.java * /hbase/branches/0.98/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/StoreFile.java * /hbase/branches/0.98/hbase-server/src/test/java/org/apache/hadoop/hbase/mapreduce/TestLoadIncrementalHFiles.java * /hbase/branches/0.98/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestHRegion.java * /hbase/branches/0.98/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/wal/TestWALReplay.java * /hbase/branches/0.98/hbase-server/src/test/java/org/apache/hadoop/hbase/util/HFileTestUtil.java

44. SUCCESS: Integrated in hbase-0.96-hadoop2 #274 (See [https://builds.apache.org/job/hbase-0.96-hadoop2/274/]) HBASE-10958 [dataloss] Bulk loading with seqids can prevent some log entries from being replayed (jdcryans: rev 1590146) * /hbase/branches/0.96/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/branches/0.96/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegionServer.java * /hbase/branches/0.96/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HStore.java * /hbase/branches/0.96/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/MemStoreFlusher.java * /hbase/branches/0.96/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/StoreFile.java * /hbase/branches/0.96/hbase-server/src/test/java/org/apache/hadoop/hbase/mapreduce/TestLoadIncrementalHFiles.java * /hbase/branches/0.96/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestHRegion.java * /hbase/branches/0.96/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/wal/TestWALReplay.java * /hbase/branches/0.96/hbase-server/src/test/java/org/apache/hadoop/hbase/util/HFileTestUtil.java
45. FAILURE: Integrated in hbase-0.96 #395 (See [https://builds.apache.org/job/hbase-0.96/395/]) HBASE-10958 [dataloss] Bulk loading with seqids can prevent some log entries from being replayed (jdcryans: rev 1590146) * /hbase/branches/0.96/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/branches/0.96/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegionServer.java * /hbase/branches/0.96/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HStore.java * /hbase/branches/0.96/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/MemStoreFlusher.java * /hbase/branches/0.96/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/StoreFile.java * /hbase/branches/0.96/hbase-server/src/test/java/org/apache/hadoop/hbase/mapreduce/TestLoadIncrementalHFiles.java * /hbase/branches/0.96/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestHRegion.java * /hbase/branches/0.96/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/wal/TestWALReplay.java * /hbase/branches/0.96/hbase-server/src/test/java/org/apache/hadoop/hbase/util/HFileTestUtil.java
46. SUCCESS: Integrated in HBase-0.98-on-Hadoop-1.1 #281 (See [https://builds.apache.org/job/HBase-0.98-on-Hadoop-1.1/281/]) HBASE-10958 [dataloss] Bulk loading with seqids can prevent some log entries from being replayed (jdcryans: rev 1590145) * /hbase/branches/0.98/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/branches/0.98/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HRegionServer.java * /hbase/branches/0.98/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/HStore.java * /hbase/branches/0.98/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/MemStoreFlusher.java * /hbase/branches/0.98/hbase-server/src/main/java/org/apache/hadoop/hbase/regionserver/StoreFile.java * /hbase/branches/0.98/hbase-server/src/test/java/org/apache/hadoop/hbase/mapreduce/TestLoadIncrementalHFiles.java * /hbase/branches/0.98/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/TestHRegion.java * /hbase/branches/0.98/hbase-server/src/test/java/org/apache/hadoop/hbase/regionserver/wal/TestWALReplay.java * /hbase/branches/0.98/hbase-server/src/test/java/org/apache/hadoop/hbase/util/HFileTestUtil.java
47. Are you waiting for me to commit [~jdcryans]? Just making sure we're not mutually waiting.
48. Sorry, went to do something else, lemme get that in (with HBASE-11008 first).
49. Now committed to 0.94 (took some time because I wanted to double check the tests I modified were all green). Thanks everyone.
50. You 'd man.
51. FAILURE: Integrated in HBase-0.94-JDK7 #132 (See [https://builds.apache.org/job/HBase-0.94-JDK7/132/]) HBASE-10958 [dataloss] Bulk loading with seqids can prevent some log entries from being replayed (jdcryans: rev 1591495) * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/HRegionServer.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/MemStoreFlusher.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/Store.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/StoreFile.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/mapreduce/TestLoadIncrementalHFiles.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/TestCompaction.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/TestHRegion.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/wal/TestWALReplay.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/util/HFileTestUtil.java
52. FAILURE: Integrated in HBase-0.94-on-Hadoop-2 #82 (See [https://builds.apache.org/job/HBase-0.94-on-Hadoop-2/82/]) HBASE-10958 [dataloss] Bulk loading with seqids can prevent some log entries from being replayed (jdcryans: rev 1591495) * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/HRegionServer.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/MemStoreFlusher.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/Store.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/StoreFile.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/mapreduce/TestLoadIncrementalHFiles.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/TestCompaction.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/TestHRegion.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/wal/TestWALReplay.java * /hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/util/HFileTestUtil.java
53. SUCCESS: Integrated in HBase-0.94 #1365 (See [https://builds.apache.org/job/HBase-0.94/1365/]) HBASE-10958 [dataloss] Bulk loading with seqids can prevent some log entries from being replayed (jdcryans: rev 1591495) * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/HRegionServer.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/MemStoreFlusher.java * /hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/Store.java *

/hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/StoreFile.java *
/hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/mapreduce/TestLoadIncrementalHFiles.java *
/hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/TestCompaction.java *
/hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/TestHRegion.java *
/hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/wal/TestWALReplay.java *
/hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/util/HFileTestUtil.java

54. FAILURE: Integrated in HBase-0.94-security #480 (See [https://builds.apache.org/job/HBase-0.94-security/480/]) HBASE-10958 [dataloss] Bulk loading with seqids can prevent some log entries from being replayed (jdcryans: rev 1591495) *
/hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/HRegion.java *
/hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/HRegionServer.java *
/hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/MemStoreFlusher.java *
/hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/Store.java *
/hbase/branches/0.94/src/main/java/org/apache/hadoop/hbase/regionserver/StoreFile.java *
/hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/mapreduce/TestLoadIncrementalHFiles.java *
/hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/TestCompaction.java *
/hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/TestHRegion.java *
/hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/regionserver/wal/TestWALReplay.java *
/hbase/branches/0.94/src/test/java/org/apache/hadoop/hbase/util/HFileTestUtil.java