Item 81
git_comments:

1. QueryParser parser = new QueryParser("contents", new WhitespaceAnalyzer()); Query query = parser.parse("handle:1");
2. make sure 2nd delete & 2nd norm "took":
3. make sure searching sees right # hits
4. add 10 docs
5. Open pre-lockless index, add docs, do a delete & * setNorm, and search
6. optimize
7. **comment:** Delete one doc so we get a .del file:
   label: documentation
8. Verifies that the expected file names were produced
9. Now verify file names:
10. Uncomment these cases & run in a pre-lockless checkout to create indices:
11. **comment:** Verify we can read the pre-XXX file format, do searches against it, and add documents to it.
    label: documentation
12. Unzips dirName + ".zip" --> dirName, removing dirName first
13. First document should be #21 since it's norm was increased:
14. make sure writer sees right total -- writer seems not to know about deletes in .del?
15. Set one norm so we get a .s0 file:
16. make sure we can do another delete & another setNorm against this pre-lockless segment:
17. open writer
18. public void testCreatePreLocklessCFS() throws IOException { createIndex("src/test/org/apache/lucene/index/index.prelockless.cfs", true); } public void testCreatePreLocklessNoCFS() throws IOException { createIndex("src/test/org/apache/lucene/index/index.prelockless.nocfs", false); }

git_commits:

1. **summary:** Lockless commits: LUCENE-701
   message: Lockless commits: LUCENE-701 git-svn-id: https://svn.apache.org/repos/asf/lucene/java/trunk@476383 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Lock-less commits
   description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the

.del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out

right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

2. **summary:** Lock-less commits
   description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check

to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

3. **summary:** Lock-less commits
   description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some

small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

4. **summary:** Lock-less commits
   description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress

testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.
label: code-design

5. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix

before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

6. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1)

did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.
label: code-design

7. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput

failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.
label: code-design

8. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress

testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.
label: code-design

9. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix

before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

10. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1)

did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

11. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1

writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

12. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in

Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

13. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably

instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

label: code-design

14. **summary:** Lock-less commits

description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest

segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

15. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when

create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.
label: code-design

16. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in

Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

17. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably

instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

18. **summary:** Lock-less commits
    description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-

dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to

1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

19. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc.

(This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

20. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening

the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

21. **summary:** Lock-less commits
description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index

for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach (originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

label: code-design

22. **summary:** Lock-less commits

    description: This is a patch based on discussion a while back on lucene-dev: http://mail-archives.apache.org/mod_mbox/lucene-java-dev/200608.mbox/%3c44E5B16D.4010805@mikemccandless.com%3e The approach is a small modification over the original discussion (see Retry Logic below). It works correctly in all my cross-

machine test case, but I want to open it up for feedback, testing by users/developers in more diverse environments, etc. This is a small change to how lucene stores its index that enables elimination of the commit lock entirely. The write lock still remains. Of the two, the commit lock has been more troublesome for users since it typically serves an active role in production. Whereas the write lock is usually more of a design check to make sure you only have one writer against the index at a time. The basic idea is that filenames are never reused ("write once"), meaning, a writer never writes to a file that a reader may be reading (there is one exception: the segments.gen file; see "RETRY LOGIC" below). Instead it writes to generational files, ie, segments_1, then segments_2, etc. Besides the segments file, the .del files and norm files (.sX suffix) are also now generational. A generation is stored as an "_N" suffix before the file extension (eg, _p_4.s0 is the separate norms file for segment "p", generation 4). One important benefit of this is it avoids files contents caching entirely (the likely cause of errors when readers open an index mounted on NFS) since the file is always a new file. With this patch I can reliably instantiate readers over NFS when a writer is writing to the index. However, with NFS, you are still forced to refresh your reader once a writer has committed because "point in time" searching doesn't work over NFS (see LUCENE-673). The changes are fully backwards compatible: you can open an old index for searching, or to add/delete docs, etc. I've added a new unit test to test these cases. All units test pass, and I've added a number of additional unit tests, some of which fail on WIN32 in the current lucene but pass with this patch. The "fileformats.xml" has been updated to describe the changes to the files (but XXX references need to be fixed before committing). There are some other important benefits: * Readers are now entirely read-only. * Readers no longer block one another (false contention) on initialization. * On hitting contention, we immediately retry instead of a fixed (default 1.0 second now) pause. * No file renaming is ever done. File renaming has caused sneaky access denied errors on WIN32 (see LUCENE-665). (Yonik, I used your approach here to not rename the segments_N file(try segments_(N-1) on hitting IOException on segments_N): the separate ".done" file did not work reliably under very high stress testing when a directory listing was not "point in time"). * On WIN32, you can now call IndexReader.setNorm() even if other readers have the index open (fixes a pre-existing minor bug in Lucene). * On WIN32, You can now create an IndexWriter with create=true even if readers have the index open (eg see www.gossamer-threads.com/lists/lucene/java-user/39265) . Here's an overview of the changes: * Every commit writes to the next segments_(N+1). * Loading the segments_N file (& opening the segments) now requires retry logic. I've captured this logic into a new static class: SegmentInfos.FindSegmentsFile. All places that need to do something on the current segments file now use this class. * No more deletable file. Instead, the writer computes what's deletable on instantiation and updates this in memory whenever files can be deleted (ie, when it commits). Created a common class index.IndexFileDeleter shared by reader & writer, to manage deletes. * Storing more information into segments info file: whether it has separate deletes (and which generation), whether it has separate norms, per field (and which generation), whether it's compound or not. This is instead of relying on IO operations (file exists calls). Note that this fixes the current misleading FileNotFoundException users now see when an _X.cfs file is missing (eg http://www.nabble.com/FileNotFound-Exception-t6987.html). * Fixed some small things about RAMDirectory that were not filesystem-like (eg opening a non-existent IndexInput failed to raise IOException; renames were not atomic). I added a stress test against a RAMDirectory (1 writer thread & 2 reader threads) that uncovered these. * Added option to not remove old files when create=true on creating FSDirectory; this is so the writer can do its own [more sophisticated because it retries on errors] removal. * Removed all references to commit lock, COMMIT_LOCK_TIMEOUT, etc. (This is an API change). * Extended index/IndexFileNames.java and index/IndexFileNameFilter.java with logic for computing generational file names. * Changed index/IndexFileNameFilter.java to use a HashSet to check file extentsions for better performance. * Fixed the test case TestIndexReader.testLastModified: it was incorrectly (I think?) comparing lastModified to version, of the index. I fixed that and then added a new test case for version. Retry Logic (in index/SegmentInfos.java) If a reader tries to load the segments just as a writer is committing, it may hit an IOException. This is just normal contention. In current Lucene contention causes a [default] 1.0 second pause then retry. With lock-less the contention causes no added delay beyond the time to retry. When this happens, we first try segments_(N-1) if present, because it could be segments_N is still being written. If that fails, we re-check to see if there is now a newer segments_M where M > N and advance if so. Else we retry segments_N once more (since it could be it was in process previously but must now be complete since segments_(N-1) did not load). In order to find the current segments_N file, I list the directory and take the biggest segments_N that exists. However, under extreme stress testing (5 threads just opening & closing readers over and over), on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. This means the listing will show a segments_N file but that file does not exist (fileExists() returns false). In order to handle this (and other such platforms), I switched to a hybrid approach

(originally proposed by Doron Cohen in the original thread): on committing, the writer writes to a file "segments.gen" the generation it just committed. It writes 2 identical longs into this file. The retry logic, on detecting that the directory listing is stale falls back to the contents of this file. If that file is consistent (the two longs are identical), and, the generation is indeed newer than the dir listing, it will use that. Finally, if this approach is also stale, we fallback to stepping through sequential generations (up to a maximum # tries). If all 3 methods fail, we throw the original exception we hit. I added a static method SegmentInfos.setInfoStream() which will print details of retry attempts. In the patch it's set to System.out right now (we should turn off before a real commit) so if there are problems we can see what retry logic had done.

jira_issues_comments:

1. ZIP file that needs to be put in src/test/org/apache/lucene/index (used by backwards compatibility test).
2. ZIP file that needs to be put in src/test/org/apache/lucene/index (used by backwards compatibility test).
3. Nice job on this very ambitious patch (all 3500 lines of it!) Good tests are certainly important! Could you elaborate on how the backward compatibility works w.r.t. modifying an old index? How do versioned norms & del files mix with older unversion files? In the absense of contention, have you noticed any performance differences in opening an IndexReader? > on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. That sucks... (but great job on the very thorough testing). Can it happen with the latest version of OS X? If not, couldn't we just require an upgrade, or do you think that other platforms suffer from this.
4. **body:** Thanks Yonik for looking at this! > Could you elaborate on how the backward compatibility works w.r.t. modifying an old index? > How do versioned norms & del files mix with older unversion files? OK, first off, the only file whose contents and name are changed is the "segments_N" file. Then, the only files whose name (but not contents) is changed are the "_X_N.del" deletes file, and "_X_N.sZ" separate norms files). Finally, the only file that is deleted is deletable. All other segments files are unchanged. The unit test I added for this (TestBackwardsCompatibility) un-zips a pre-lockless index (I have one zip file for CFS format and one for non-CFS) and then runs various tests: adding docs, deleting a doc, setting a norm for a doc, searching, etc., verifying that the results are then as expected. On opening an pre-lockless index, first we see only a "segments" file with no _N extension, and record its generation as 0. Second, since that file's contents doesn't lead with format lockless code (-2), we know to load the old segments info format that does not contain the "del" gen, "separate norm" gen nor the "isCompundFile" marker. When loading each Segmentinfo, since the format of the segments files was old, we record this with "preLockless = true" and set delGen and isCompoundFile to 0 and normGen to null. 0 means "must check filesystem for existence". The various methods (hasSeparateNorms, hasDeletions, etc.) know to handle these "0" cases as falling back to filesystem existence checks of the existing naming (ie, _X.del). I tried to capture / contain all of this inside the methods of SegmentInfo. Now, when a writer commits to this pre-lockless index, we write in the new (format lockless) format and to the file "segments_1". (Actually, with compound file turned on, we then make a .cfs file and commit again to "segments_2"). This file will reference all of the old segments (except eg those deleted by a merge) plus the one new segment. The old segments have written the "0"'s for delGen, normGen (null is written as -1 length), isCompoundFile so that on re-loading this segments_2 file these segments remain pre-lockless. The SegmentInfo for the new segment file will have isCompoundFile=1 (meaning it is a compound file), delGen=-1 (there are no separate deletes yet) and normGen=null (there are no separate norms yet). When normGen is null, we look at "preLockless" to differentiate whether this means there are no separate norms at all for any fields, or, this segment is pre lockless and therefore we must fallback to the filesystem check. If a delete or setNorm call takes place against an old segment format, we will at that time create "generation 1" for that file. This means you can have an old segment whose separate del file is still "generation 0" (you have to check for existence of _X.del) but whose separate norms generations are known (or, only certain fields are known and the others are "generation 0" and require filesystem check). This means an "old" segment file could become "slightly newer" as norm/del changes are made against it. So an index can have mixed old/new segments. The SegmentInfo for each segment keeps track of old/new (and tries to hide these implementation details under its methods) with delGen, normGen, isCompoundFile and preLockless (which is derived from isCompoundFile). Once an optimize is done, or, all old segments have been merged away, then all segments are now the lockless format.
   label: code-design
5. > > on one platform (OS X) I found that the directory listing can be incorrect (stale) by up to 1.0 seconds. > > That sucks... (but great job on the very thorough testing). > Can it happen with the latest version of OS X? If not, couldn't we just require an upgrade, or do you think that other platforms suffer from this.

Yes, this was very surprising and annoying! Unfortunately, this happens on the latest version of OS X (10.4.8). Other platforms that I tested do "the right thing" meaning when you list a directory the NFS client on Linux first checks with the server to see if its cache is stale and correctly clears the cache. Windows SMB is also always correct. But I think it's entirely likely that filesystems do this kind of time-based (only) cache validation. I figured better safe than sorry here, and Lucene should tolerate stale caching around either file contents or directory listing.

6. **body:** > In the absense of contention, have you noticed any performance differences in opening an IndexReader? [First one side note: it's during contention that lock-less really shines. Because, currently Lucene hits at least a [default] 1.0 second delay under contention, whereas lockless immediately retries. And, readers now have contention with one another, but not with lock-less.] I ran the following basic performance test. In each case I measure avg wall clock time to instantiate readers & writers against the same index. A writer creates the index, adding documents as quickly as it can, and commits and closes/reopens its writer every 2 seconds. Then, a reader reads against the same index, just instantiating a searcher then closing it and then pausing for 2 seconds. I skew the writer by 1 sec to try to minimize contention. Each test is avg of 3 runs, where each run is 2 minutes. In order to not count contention, I made temporary mods, to both current lucene & lockless, to throw IOException on hitting contention. Then, I catch that above and discard the time for that one instantiation of reader or writer. All times are mili-seconds. Each time is formatted as current Lucene time followed by lockless in (...)s: Local index (Linux): 4.62 (6.04) Local index (WIN32): 85.45 (66.37) NFS remote index (Linux): 171.26 (11.04) SMB remote index (WIN32): 48.91 (31.55) The "remote index" case means a writer on that OS, and a reader on another machine with the same OS, reading the index on a mount from the writer machine (ie, writer is writing locally and reader is reading remotely). One caveat: I saw quite a bit of variance between runs. I tried to eliminate causes (stopped all services, other applications, etc.) but still there is variance. Maybe I should be taking the "minumum" time seen instead of the average (this was mentioned on the benchmarking Jira issue)... Anyway, the surprising thing is that lockless is faster in most cases In the remote cases (especially NFS) it's quite a bit faster. I think this may be because lockless does far fewer "fileExists" calls compared to current Lucene. For example, the "openNorms" call presently does a "fileExists" call per field that has norms index times the number of segments. I'm not sure these speedups/differences are all that important in a typical Lucene use case, where the cost of instantiating a reader is amortized over all the searches that occur during the lifetime of that reader. Though, maybe one real difference is: if we can make sure the latency is low enough, it's OK to have a single query pay the price of reopening the searcher. Ie, it becomes reasonable to have an incoming query first check whether the searcher is current and if not, reopen it, and then run the query, vs having separate background thread do this, which is certainly feasible just more complicated.
   label: code-design
7. **body:** Thanks for all the details Michael! A few more random comments and questions: In the future, it might be nice if there was an option to disable segments.gen to be more friendly to write-once filesystems like HDFS. As far as performance goes, I was personally interested in the contentionless case since that's what processes that coordinate everything (like Solr) will see. I'm not sure I understand the "segments.gen" logic of writing two longs that are identical. Looking at the code, it doesn't seem like you are implementing this: http://www.nabble.com/Re%3A-Lock-less-commits-p5978090.html Are there two longs instead of one in order to leave "space" for that implementation if needed, w/o having to change the file format? The file deleting code does much more than in the past, and that's a good thing IMO. For example it looks like leftover non-compound segment files from a failed CFS merge (say the JVM dies) will now be cleaned up! I'm having a hard time figuring out how older delete files are removed (since they contain the current segment name, it looks like findDeletableFiles would skip them).
   label: code-design
8. **body:** Good questions! > In the future, it might be nice if there was an option to disable > segments.gen to be more friendly to write-once filesystems like > HDFS. I think this makes sense. I will add control over this on the next iteration of the patch. > As far as performance goes, I was personally interested in the > contentionless case since that's what processes that coordinate > everything (like Solr) will see. Ahh got it, OK. That's fair. > I'm not sure I understand the "segments.gen" logic of writing two > longs that are identical. Looking at the code, it doesn't seem like > you are implementing this: > http://www.nabble.com/Re%3A-Lock-less-commits-p5978090.html > Are there two longs instead of one in order to leave "space" for > that implementation if needed, w/o having to change the file format? Right, I settled on a simplification of that approach. I write two longs so that reader can read both & compare and only trust them if they are identical. With one long I was worried eg that perhaps 3 bytes from the writer were written but not yet the remaining 5 bytes, and then reader would get the wrong value. I don't think IO systems guarantee atomicity of eg 8 byte chunks (though in practice it's probably often the case).

One thing I will also add is a version header to this file. > The file deleting code does much more than in the past, and that's a > good thing IMO. For example it looks like leftover non-compound > segment files from a failed CFS merge (say the JVM dies) will now be > cleaned up! Oh, right! In fact any time an index crashes not having committed its segments file, there is potential for leftover (unreferenced) files now. This separate IndexFileDeleter class should correctly reclaim in such cases. And even other potential future situations like the discussion in LUCENE-702 would be reclaimed correctly with this approach. > I'm having a hard time figuring out how older delete files are > removed (since they contain the current segment name, it looks like > findDeletableFiles would skip them). Oooh -- you are correct. I do record this file for deleting at the point it becomes unreferenced (ie, as a reader is committing its separate norms/deletes), and then I delete this file after the commit is done. But if JVM crashes after the new del file was written and before the commit, then you're right on restarting I don't correctly delete the unreferenced old _X_N.del files, nor I believe the separate norms _X_N.sM files. I will add a unit test to verify this bug and then fix it -- good catch!
label: code-design

9. Can the following scenario happen with lock-less commits? 1 A reader reads segments.1, which says the index contains seg_1. 2 A writer writes segments.2, which says the index now contains seg_2, and deletes seg_1. 3 The reader tries to load seg_1 and fails.
10. > 3 The reader tries to load seg_1 and fails. That wouldn't be considered a failure because it's part of the retry logic. At that point, an attempt would be made to open seg_2. To minimize the possibility of this happening, the segments are opened in reverse order (since the last segments change the most often). Then a question might be, could a writer possibly change the index fast enough to prevent a reader from opening at all? I don't think so (and it would be a mis-configured writer IMO), but maybe Michael could speak to that.
11. > That wouldn't be considered a failure because it's part of the retry logic. At that point, an attempt would be made to open seg_2. From the description of the retry logic, I thought the retry logic only applies to the loading of the "segments_N" file, but not to the entire process of loading all the files of an index. You are right, it wouldn't be a failure if the retry logic is applied to the loading of all the files of an index.
12. > > In the future, it might be nice if there was an option to disable > > segments.gen to be more friendly to write-once filesystems like > > HDFS. > I think this makes sense. I will add control over this on the next > iteration of the patch. Just to be clear, I didn't mean that I thought it was needed now... there is another place in Lucene that prevents write-once from working (segment file lengths at the beginning IIRC). When this option is added, perhaps the configuration name should be generic and not tied to the implementation specifics that could change more frequently? Something like WRITE_ONCE or setWriteOnce()?
13. **body:** Right, this is just normal contention. We do indeed retry around not only loading of segments_N but also the loading of the individual segments files. There are other places (eg lastModified()) that do other things with the segments file. These places also use the retry logic. In Lucene currently, contention causes a pause (default 1.0 second) and then retry to obtain the commit lock. With lockless, we simply retry immediately loading the latest segments_N file. It's important to note that at any given instant, the index is always "consistent" (well, except for issues like LUCENE-702). But, because a reader takes non-zero time to load the index, you can hit contention if a writer's commit spans that time. If a reader could load an index in zero time there would never be contention. There are several ways that contention will manifest itself. These are just the different alignments / convolutions of the series of steps that reader goes through "sliding against" the series of steps that a writer goes through: * Reader opens the segments_N but in reading it hits EOF because writer has not finished writing it yet. * Reader opens segments_N, fully reads its contents, but then hits IOException on loading each segment file because during this team writer has committed and is now deleting segments files. This case is your example above. * Reader opens segments_N, but hits IOException while reading its contents because it was deleted by writer before reader could read all of its contents (should only happen on fileystems that don't do "delete on last close" or "can't delete open files"). * Reader takes listing of directory, locates segments_N, but fails to open that file because writer has now removed it. Anyway, on hitting an IOException, we first retry segments_N-1 (if it exists). Failing that we recheck the directory for latest segments_N. If N has advanced we try that. If N has not advanced we give it one more chance to load (since it could be on first try we hit case 1 above). If it fails that second chance and on re-listing we are still at N, we throw the original exception we hit. I added a couple of tests cases to TestIndexWriter to verify that a messed up index indeed throws an IOException. On Yonik's question: > Then a question might be, could a writer possibly change the index > fast enough to prevent a reader from opening at all? I don't think > so (and it would be a mis-configured writer IMO), but maybe Michael > could speak to that. This is definitely possible. This really is a form of "starvation". If a writer is committing too fast, or, readers are constantly re-opening too fast, they will

starve one another. Both current Lucene and lockless will hit starvation under high enough rate of commit/opens, but, different things happen. EG LUCENE-307 issue is exactly this case on the current Lucene. Lockless will retry indefinitely though may at some point succeed (but take many retries to do so). Still, I think the point at which starvation starts to happen is far beyond a normal usage of Lucene (ie, committing > ten times / sec).
label: code-design

14. > > In the future, it might be nice if there was an option to disable > > segments.gen to be more friendly to write-once filesystems like > > HDFS. > I think this makes sense. I will add control over this on the next > iteration of the patch. > Just to be clear, I didn't mean that I thought it was needed now... > there is another place in Lucene that prevents write-once from > working (segment file lengths at the beginning IIRC). > When this option is added, perhaps the configuration name should be > generic and not tied to the implementation specifics that could change > more frequently? Something like WRITE_ONCE or setWriteOnce()? OK, I see, this is part of a wider context. Maybe it's the creation of the compound file you're thinking of? That writes 0's into the header, adds the files, then rewinds and puts the actual offsets into it. Then let's open a separate issue to track this -- I'll do that. Don't want to make this patch any bigger!
15. **body:** OK, another version of the lockless commits patch with these fixes: - Added new unit test (TestIndexFileDeleter) for deleter, caught the above bug Yonik found (we can fail to delete orphan'd separate del/norm files), fixed it, and unit test now passes. - We were also failing to deleted orphan'd .fN files (norm files that do get included into CFS file). Fixed that case too. - Added version header to segments.gen file. - Added static setter/getters for advanced configuration of the retry logic. Note: this required making the SegmentInfos class public. You still need to put those two zip files into src/test/org/apache/lucene/index after applying this patch. This addresses all feedback/TODOs that I knew about. All unit tests pass.
label: code-design
16. Looks good Michael, I think this is ready to commit! Does anyone have any concerns with this going into the trunk?
17. I'm all for the patch ... the only thing I'm wondering is about release timing, if that's issue? This changes the on-disk format, which affects more than Lucene Java and, should anyone that's using Lucene out there care (via scripts, etc.), the naming of files on disk. I'm just wondering if there's any interest/reason for doing a 2.1 before something with those side effects?
18. Steven - I don't see any issues with this going in before 2.1. As a matter of fact, this may be a sufficiently substantial change that will make us want to make a 2.1 release. Maybe Michael should commit this next week.
19. Oooh -- I would love to!
20. Closing all issues that were resolved for 2.1.