Item 84
git_comments:

1. skip over the part that we aren't loading
2. field name
3. this will happen if the field is not compressed
4. Disable reading raw docs in 2.x format, because of the removal of compressed fields in 3.0. We don't want rawDocs() to decode field bits to figure out if a field was compressed, hence we enforce ordinary (non-raw) stored field merges for <3.0 indexes.
5. * @deprecated Only kept for backward-compatbility with <3.0 indexes. Will be removed in 4.0.
6. uncompress the value and add as string
7. * @deprecated Kept for backwards-compatibility with <3.0 indexes; will be removed in 4.0
8. Lucene 3.0: Removal of compressed fields
9. **comment:** This was used in 2.9 to generate an index with compressed field: static final int TEXT_COMPRESSED_LENGTH; static { try { TEXT_COMPRESSED_LENGTH = CompressionTools.compress(TEXT_TO_COMPRESS.getBytes("UTF-8")).length; } catch (Exception e) { throw new RuntimeException(); } }
   label: code-design
10. check if field was decompressed after optimize
11. read the size from the binary value using DataInputStream (this prevents us from doing the shift ops ourselves):
12. This was used in 2.9 to generate an index with compressed field: static final int BINARY_COMPRESSED_LENGTH = CompressionTools.compress(BINARY_TO_COMPRESS).length;
13. look into sub readers and check if raw merge is on/off
14. FieldSelectorResult.SIZE returns compressed size for compressed fields, which are internally handled as binary; do it in the same way like FieldsWriter, do not use CompressionTools.compressString() for compressed fields:
15. FieldSelectorResult.SIZE returns 2*number_of_chars for String fields:
16. test that decompression works correctly
17. This was used in 2.9 to generate an index with compressed field: if (id % 2 == 0) { doc.add(new Field("compressed", TEXT_TO_COMPRESS, Field.Store.COMPRESS, Field.Index.NOT_ANALYZED)); doc.add(new Field("compressedSize", Integer.toString(TEXT_COMPRESSED_LENGTH), Field.Store.YES, Field.Index.NOT_ANALYZED)); } else { doc.add(new Field("compressed", BINARY_TO_COMPRESS, Field.Store.COMPRESS)); doc.add(new Field("compressedSize", Integer.toString(BINARY_COMPRESSED_LENGTH), Field.Store.YES, Field.Index.NOT_ANALYZED)); }

git_commits:

1. **summary:** LUCENE-1960: Add support for reading Field.Store.COMPRESS fields again and decompress on merge/optimize. Also convert to a new stored file version number.
   message: LUCENE-1960: Add support for reading Field.Store.COMPRESS fields again and decompress on merge/optimize. Also convert to a new stored file version number. git-svn-id: https://svn.apache.org/repos/asf/lucene/java/trunk@829972 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Remove deprecated Field.Store.COMPRESS

description: Also remove FieldForMerge and related code.

jira_issues_comments:

1. All core & contrib tests pass.
2. Committed revision 822978.
3. **body:** Just one question: What happens with indexes that already had compressed fields. Do they behave as before? *edit* From the patch I see that you also removed the bitmask for testing compression in index format. So an index with compressed fields from 2.9 should behave undefined. In my opinion, support for reading compressed fields should stay available, but not for writing. And: as soon as segments are merged, they should get silently uncompressed (what should be no problem if the special FieldForMerge is no longer used). Also the constant bitmask for compression should stay "reserved" for futrure use.
   label: code-design
4. Users can use CompressionTools#decompress() now. They just must know now which binary fields are compressed. I don't think the SegmentMerger should uncompress automatically? That'd make <=2.9 indexes suddenly bigger.
5. {quote} Also the constant bitmask for compression should stay "reserved" for futrure use. {quote} Yeah I think you're right, we must make sure that we don't use this bit for something else, as old indexes might have it set to true already. I'll add it back with a deprecation comment saying that we'll remove it in 4.0. (4.0 won't have to be able to read <3.0 indexes anymore).
6. Reopening so that I don't forget to add back the COMPRESS bit.
7. In the discussion with Mike, we said, that all pre-2.9 compressed fields should behave as before, e.g. they should automatically decompress. It should not be possibile to create new ones. This is just index compatibility, in the current version it is simply not defined what happens with pre-2.9 fields. The second problem are older compressed fields using the modified Java-UTF-8 encoding, which may not correctly decompress now (if you receive with getByte()) The problem with your patch: If the field is compressed and you try to get it, you would not hit it (because it is marked as String, not binary). The new self-compressed fields are now should be "binary", before they were binary *or* string. See the discussion in LUCENE-652
8. {quote} The problem with your patch: If the field is compressed and you try to get it, you would not hit it (because it is marked as String, not binary). The new self-compressed fields are now should be "binary", before they were binary or string. See the discussion in LUCENE-652 {quote} Hmm I see. I should have waited a bit with committing - sorry! I'll take care of it tomorrow, it's getting too late now.
9. **body:** No problem. :-) I think it should not be a big task to preserve the decompression of previously compressed fields. Just revert FieldsReader changes and modify a little bit.
   label: code-design
10. **body:** bq. I don't think the SegmentMerger should uncompress automatically? That'd make <=2.9 indexes suddenly bigger. If we re-add support for compressed fields, we have to also provide the special FieldForMerge again. To get rid of this code (which is the source of the problems behind the whole COMPRESS problem), we could simply let it as it is now without FieldForMerge. If you merge two segmets with compressed data then, without FieldToMerge it gets automatically decompressed and written in uncompressed variant to the new segment. As compressed fields are no longer supported, this is the correct behaviour. During merging, the compress bit must be removed. The problem are suddenly bigger indexes, but we should note this in docs: "As compressed fields are no longer supported, during mering the compression is removed. If you want to compress your fields, do it yourself and store it as binary stored field." Just for confirmation: I have some indexes with compress enabled (for some of the documents, since 2.9 we do not compress anymore, newly added docs have no compression anymore [it was never an good idea because of performance]). So I have no possibility to get these fields anymore, because I do not know if they are compressed and cannot do the decompression myself. For me, this data is simply lost. I think Solr will have the same problem.
    label: code-design
11. Adds the compress bit back to FieldsWriter and the uncompress code to FieldsReader. Uwe, could you review the patch?
12. I will look into this tomorrow morning. I am to tired now, have to go to bed. I will also check the implications of not having the FieldForMerge.
13. Sorry, I forgot this one, will check tomorrow with some old indexes using compressed fields.
14. I created an index with some compressed binary and String fields with 2.4 and verified that it gets decompressed correctly. The test fails currently on trunk (as expected) and passes with the latest patch. However, there's one issue here: the compressed field gets silently uncompressed during merge, *only* if

in the less efficient merge mode that doesn't use FieldsReader#rawDocs() and FieldsWriter#addRawDocuments(). So now this doesn't sound like a great solution that we sometimes uncompress the fields automatically and sometimes don't. I think we have three options: 1. Change FieldsWriter#addRawDocuments() to uncompress on-the-fly 2. Revert the FieldForMerge changes too and never uncompress automatically during merge 3. Make it possible for the user to uncompress fields with CompressionTools, no matter which UTF format the data was stored with I don't really want to do 1., because it will have a performance impact for all fields (you have to look at the field bits even in raw merge mode). With 2. we will have to keep most of the compress/uncompress code in Lucene until 4.0, we'll just not make it possible anymore to add Store.COMPRESS fields with 3.0 (that's already how trunk is). For 3. we'd have to add a deprecated isCompressed() method that the user can call.

15. How shall we proceed here? (see my previous comment)
16. **body:** I still prefer 1, but maybe it's not so good. Else I would implement 2 (even if we need FieldForMerge). Just remove the COMPRES flag that nobody can add any compressed fields anymore. 3 is bad, because it needs you to change your code on the change between 2.9 and 3.0 if you had compressed fields. In 2.9 they were automatically uncompressed, in 3.0 not. This would make it impossible to replace the lucene jar (which is currently possible if you remove all deprecated calls in 2.9).
    label: code-design
17. If we want to stay with the current patch, we place a warning that indexes can suddenly get bigger on merges. We note this in changes.txt. If one wants to regenerate the index with the stored fields decompressed, he could simply use the IndexSplitter contrib module recently added. This command line tool uses addIndexes and therefore merges all segments into a new index. With option 1, they get decompressed. If somebody wants real compressed fields again, he has to write code and reindex using CompressableStringTools.
18. **body:** I'm actually -1 for option 1). The whole implementation of addRawDocuments() would have to change, and the necessary changes would kind of defeat its purpose. If we do 2) nobody will be able to use an index that has compressed fields in 4.0 anymore, and to convert it they have to manually reindex (which might not always be possible). Of course our policy says that 4.0 must not be able to read <3.0 indexes anymore, however normally users can take a 2.x index, optimize it with 3.x, and then 4.0 can read it without problems. This wouldn't be possible with 2).
    label: code-design
19. And how about keeping the current lucene-1960-1.patch? It works for me as I exspected. The only problem is that we do not decompress the fields for sure on optimizing?
20. Can't we detect that we're dealing w/ an older version segment and not use addRawDocuments when merging them (and uncompress when we merge)?
21. **body:** Right, because FieldsReader#rawDocs() does not decode the field bits, so it doesn't know which fields are compressed. If we want to change that it would have a significant negative performance impact on *all* stored fields.
    label: code-design
22. Good idea, from where take the version? Or better, we look into the FieldInfos of the segment-to-merge and look if there is the compressed flag set for one of the fields. If yes, do not use addRawDocuments. It there the possibility to see this flag also or'ed segment-wise (like a field is omitNors is per-segment)?
23. {quote} Can't we detect that we're dealing w/ an older version segment and not use addRawDocuments when merging them (and uncompress when we merge)? {quote} So then any 2.x index (including 2.9) would not be merged in the optimized way with 3.x. I'm actually not even sure how much of a slowdown this is. Did you (or anyone else) ever measure that?
24. But this is only one-time. As soon as it is optimized it is fast again. Because of that I said, one could use a tool to enforce optimization or the new IndexSplitter can also do the copy old to new index.
25. **body:** {quote} Or better, we look into the FieldInfos of the segment-to-merge and look if there is the compressed flag set for one of the fields. {quote} For a second earlier I had the same idea - it would be the most convenient solution. BUT: bummer! no compressed flag in the fieldinfos... It's a bit per stored field *instance*.
    label: code-design
26. {quote} But this is only one-time. As soon as it is optimized it is fast again. Because of that I said, one could use a tool to enforce optimization or the new IndexSplitter can also do the copy old to new index. {quote} That's right, I'm just trying to make sure we all understand the consequences. Would be nice to know how much longer it takes though. If everyone else is ok with this approach I can work on a patch.
27. So the idea is to raise the version number of the stored fields file by one in 3.0. All new or merged segments get this version number? When merging, for all versions before the actual one we do not use addRawDocuments() when copying contents. The current lucene-1960-1.patch stays unchanged.

28. Yes, I believe this would work.
29. Then +1 from me!
30. I have some large indexes here from 2.9 with compressed XML documents in stored fields. I can compare the optimization time for Lucene 2.9 and Lucene 3.0 with your patch.
31. **body:** It was as easy as changing this method in FieldsReader: {code:java} boolean canReadRawDocs() { // Disable reading raw docs in 2.x format, because of the removal of compressed // fields in 3.0. We don't want rawDocs() to decode field bits to figure out // if a field was compressed, hence we enforce ordinary (non-raw) stored field merges // for <3.0 indexes. return format >= FieldsWriter.FORMAT_LUCENE_3_0_NO_COMPRESSED_FIELDS; } {code} Uwe, I made some quick tests and it looks good. But I don't have any indexes with compressed fields (we don't use them), so I'll wait for you to test it out with your indexes that you mentioned.
    label: test
32. **body:** Attached is my comparison of an unoptimzed Lucene 2.9 index optimized with 2.9 and optimized with 3.0 with the latest patch. The index was about 5.7 GB big and contained 4 compressed stored fields per document (in addition to ther fields uncompressed) containing XML documents. After optimization with 3.0, the size doubled (which is because of the very good compression of XML documents). The optimization took about double time with 3.0, because the fields were decompressed and no addRawDocuments was used. To confirm, that everything worked normal after this initial optimization, I updated 38 documents in both indexes and optimized again. The optimization time of 2.9 was identical, 3.0 took a little bit longer, which is because the uncompressed field created more copy i/o, which you see in the "time" output as system time. User time during the initial 2.9 -> 3.0 optimization was much larger. The attached document contains all numbers together with the checkindex outputs before and after each step.
    label: code-design
33. I forgot the infos about the used system: Sun X4600 Server, Solaris 10 x64, 16 Cores, 32 GB RAM, 64 bit JVM 1.5.0_21, -Xmx1512M, RAID 5
34. **body:** I do not know if this is a bug in 2.9.0, but it seems that segments with all documents deleted are not automatically removed: {noformat} 4 of 14: name=_dlo docCount=5 compound=true hasProx=true numFiles=2 size (MB)=0.059 diagnostics = {java.version=1.5.0_21, lucene.version=2.9.0 817268P - 2009-09-21 10:25:09, os=SunOS, os.arch=amd64, java.vendor=Sun Microsystems Inc., os.version=5.10, source=flush} has deletions [delFileName=_dlo_1.del] test: open reader.....OK [5 deleted docs] test: fields.....OK [136 fields] test: field norms.....OK [136 fields] test: terms, freq, prox...OK [1698 terms; 4236 terms/docs pairs; 0 tokens] test: stored fields.....OK [0 total field count; avg ? fields per doc] test: term vectors.....OK [0 total vector count; avg ? term/freq vector fields per doc] {noformat} Shouldn't such segments not be removed automatically during the next *commit*/close of IndexWriter? But this would be another issue. In my opinion, we are fine with the current approach, the longer optimization time is rectified by the larger index size because of no compression anymore and the more heavyer initial merge without addRawDocument is only 30% slower (one time!). +1 for committing
    label: code-design
35. bq. I do not know if this is a bug in 2.9.0, but it seems that segments with all documents deleted are not automatically removed Lucene doesn't actually short-circuit this case, ie, if every single doc in a given segment has been deleted, it will still merge it [away] like normal, rather than simply dropping it immediately from the index, which I agree would be a simple optimization. Can you open a new issue? I would think IW can drop such a segment immediately (ie not wait for a merge or optimize) on flushing new deletes.
36. Will do! -> LUCENE-2010
37. **body:** For this case: Should we add some testcase in TestBackwardsCompatibility, that tests, that pre 3.0 indexes with compressed fields are correctly uncompressed on optimize and also can be correctly read? I do not know how to do this and if the current BW test indexes in the zip files contain compressed fields.
    label: test
38. bq. Should we add some testcase in TestBackwardsCompatibility +1, that'd be great What I usually do is make a mod that creates compressed fields, in the prior release (2.9.x), then uncomment the two methods that create new back compat indexes, zip them up, carry them forward to trunk, and modify the trunk test to test them. Best to also carry forward the code that generated the back compat index, though in this since COMPRESS is removed, you'll have to comment it out w/ a comment stating "this was used in 2.9 to create field XXX".
39. I am working on it. I already patched the 2.9 version and created the test index. In 3.0 I use if dirName.startsWith("29.") to only do the optimize tests for this index and no other version.

40. Modified patch with testcase for backwards compatibility. It was a little bit trick to check if a field was actually compressed in the source index, but it worked with FieldSelectorResult.SIZE and a test-compression/chars*2. The test was done before/after optimize and it is verified that before the field was still compressed (even it is larger in reality, harhar) and uncompressed after optimize. Should I commit the index creation to the current 2.9 branch or not? I can commit this, if everybody is happy (because I have the zip files already in my checkout added). Or will you do it, Michael?
41. bq. Should I commit the index creation to the current 2.9 branch or not? +1, and ideally also, commented out, in trunk
42. OK. I will post new test indexes and a new patch, because I want to also test binary stored fields. I will do it by creating a binary/string field for id%2==0 or 1. The patch for the index creating must be committed to 2.9 branch, not the backwards tests (because COMPRESS is undefined there, too!)
43. Here updated patch and ZIP index files. Now also binary fields are tested in the same way. Even document ids get a string compressed field, odd document ids a binary one. Also attached is the patch for the 2.9 branch (not BW branch!!!). In trunk, the index creation is commented out, it's just there for reference.
44. Small optimization.
45. An additional check, that the raw merge mode is disabled for all segments in 2.9 index and enabled after optimizing with 3.0.
46. After thinking a little bit about it: Is it ok to test the size of the compressed field by recompressing it with another target VM? E.g., maybe I created the test 2.9 index with another Java Version (1.5.0_21) where the deflate function is a little bit different implemented and so the test in 3.0 will fail, because maybe someone with Java 6 ran the test using another libzip? In this case, I would add another stored field in the test index, that contains the length of the compressed data during creation of the index in the source VM, to be checked with FieldSelectorResult.SIZE? Opinions?
47. Attached the new indexes and patch that stores the compressed size together with the compressed value as stored field in the 2.9 index. This is now secure and invariant on different in-JDK compression implementations.
48. **body:** Add an assertion in FieldsReader that checks, that 3.0 indexes have no compressed fields. Also a small test cleanup. Its ready to commit now!
    label: code-design
49. {quote} I can commit this, if everybody is happy (because I have the zip files already in my checkout added). Or will you do it, Michael? {quote} Go ahead! Cool that you added the bw-test. I've been wanting to do that too, but I didn't have time yet. Thank you!
50. OK, will do it a later in the evening (MEZ), have no time now. I will also add a good changes.txt entry in behaviour change or like that.
51. Committed revision 829972. Thanks also to Michael Busch!