Item 83

git_comments:

git_commits:

1. **summary:** LUCENE-2621: SegmentMerger returns MergeState
   message: LUCENE-2621: SegmentMerger returns MergeState git-svn-id: https://svn.apache.org/repos/asf/lucene/dev/branches/lucene2621@1198728 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Extend Codec to handle also stored fields and term vectors
   description: Currently Codec API handles only writing/reading of term-related data, while stored fields data and term frequency vector data writing/reading is handled elsewhere. I propose to extend the Codec API to handle this data as well.

jira_issues_comments:

1. .. and norms while we're at it.
2. bq. .. and norms while we're at it. just for the record we should explore if norms could just be a doc payload see LUCENE-2186
3. FYI - I marked this as gsoc-2011 I think this would make a wonderful project. If any student needs more info go ahead and ask on the dev list.
4. Brief Summary for GSoC Students: This issue is about extend Codec to handle also stored fields and term vectors This is a very interesting and at the same time very much needed feature which involves API Design, Refactoring and in depth understanding of how IndexWriter and its internals work. The API which needs to be refactored (Codec API) was made to consume PostingLists once an in memory index segment is flushed to disc. Yet, to expose Stored Fields to this API we need to prepare it to consume data for every document while we build the in memory segment. So there is a little paradigm missmatch here which needs to be addressed.
5. Brief Summary for GSoC Students: This issue is about extend Codec to handle also stored fields and term vectors This is a very interesting and at the same time very much needed feature which involves API Design, Refactoring and in depth understanding of how IndexWriter and its internals work. The API which needs to be refactored (Codec API) was made to consume PostingLists once an in memory index segment is flushed to disc. Yet, to expose Stored Fields to this API we need to prepare it to consume data for every document while we build the in memory segment. So there is a little paradigm missmatch here which needs to be addressed.
6. I've some questions here to make sure I'm looking into the correct codes: When you mentioned Codec API, do you mean the abstract class org.apache.lucene.index.codecs.Codec? Term vectors refer to org.apache.lucene.index.TermFreqVector, and it is processed by TermVectorsWriter now, correct? But what are the stored fields? I cannot find them immediately. BTW, is there any design document of Lucene in the Wiki?
7. bq. When you mentioned Codec API, do you mean the abstract class org.apache.lucene.index.codecs.Codec? Yes that is the main entry point. Currently a codec offer a FieldsConsumer which is pulled by the IndexWriter upon a flush request. Codecs are assigned per field and segment via the CodecProvider. So each field can have its own codec and each codec can have a different implementation. Yet, currently we only provide codec support for the reverse index so a codec can customize the term dictionary (TermsEnum would be the API counterpart) and posting lists

(DocsEnum / DocsAndPositionsEnum in the API). What this issue tries to do is to open up this API as a general low level customization layer that enables users to also customize how Stored Fields and TermVectors are stored on disk. bq. Term vectors refer to org.apache.lucene.index.TermFreqVector, and it is processed by TermVectorsWriter now, correct? yes thats true. bq. But what are the stored fields? I cannot find them immediately. there should be a StoredFieldsWriter and a FieldsReader. bq. BTW, is there any design document of Lucene in the Wiki? nothing that I would call a design document. there are some pages which could be similar to what you are looking for but those might be out of date. You should maybe look int the corresponding issues to find design decisions.

8. I assigned this to me since I would volunteer to mentor this issue. Hope that is ok Andrzej?
9. Sure, go ahead.
10. **body:** Here is a minimal 'rote refactor' for stored fields, there is a lot more to do (e.g. filenames/extensions should come from codec, segmentmerger optimizations (bulk merging) should not be in the API but customized by the codec, the codec name (format) of fields should be recorded in the index, we should implement a simpletext version and refactor/generalize, ...) but more importantly, I think we need to restructure the class hierarchy: Codec is a per-field thing currently but I think the name "Codec" should represent the entire index... maybe what is Codec now should be named FieldCodec? maybe the parts of CodecProvider (e.g. segmentinfosreader, storedfields, etc) should be be moved to this new Codec class? in this world maybe PreFlex codec for example returns its hardcoded representation for every field since in 3.x this stuff is *not* per field, and with more of the back compat code refactored down into PreFlex. Would be good to come up with a nice class naming/hierarchy that represents reality here.
    label: code-design
11. **body:** Awesome! I think, like the postings, we can add a .merge() method, and the impl for that would do bulk-merge when it can? On the restructuring, maybe we can go back to a PerFieldCodecWrapper, which is itself a Codec? This would simplify CodecProvider back to just being a name -> Codec instance provider? We would still use SegmentCodecs/FieldInfo(s) to compute/record the codecID, though, in theory this could become "private" to PFCW once it's a Codec again.
    label: code-design
12. **body:** I think so, assuming FieldInfos etc are *also* read/written by the codec. Then I think PFCW could be an abstract class that writes per-field configuration into the index, but for example PreFlexCodec would *not* extend this class, as a 3.x index is the same codec across all fields. I think if we do things this way we have a lot more flexibility with backwards compatibility instead of all this if-then-else conditional version-checking code when reading these files... Really, for example if someone wanted to make a Codec that reads a lucene 2.x indexes (compressed fields and all) they should be able to do this if we reorganize this right.
    label: code-design
13. I created a branch (https://svn.apache.org/repos/asf/lucene/dev/branches/lucene2621) for extending and refactoring the codec API to cover more portions of the index... I think it would be really nice to flesh this out for 4.0
14. CodecProvider -> Codec and Codec -> FieldCodec makes sense to me. This way the Codec would be responsible for all global index parts (seginfos, fieldinfos), and it would provide API to manage per-field data (FieldCodec), stored fields (FieldsWriter/Reader, perhaps a class to tie these two together), and term vectors (TermVectorsWriter/Reader, again grouped in a class). Re. current patch: this looks like a great start. I discovered a problem lurking in SegmentMerger.mergeFields. setMatchingSegmentReaders checks only fieldInfos for compatibility, but it doesn't check the codecs (and fieldWriter/fieldReader) compatibility. It's happy then to use the matchingSegmentReader directly, which results in raw documents encoded with one codec being sent as a raw stream to a fieldWriter from another codec. Also, SegmentInfo.files() is messy, it should be populated from codecs - as soon as I changed the extension of the stored fields file things exploded because TieredMergePolicy couldn't find the .fdx file reported by files().
15. yeah another alternative name to FieldCodec would be something like PostingsFormat (or similar). Because there is a big difference between PreFlex (which is actually a codec), and Memory/Pulsing. as far as the current patch: yeah its missing a lot... because as soon as I started digging here I thought, well we have to probably try to fix these codec classes first before going further. Really I would like for some of the stuff like shared docstores to be private to PreFlex codec, part of fixing the files() issue. Same with the merging, this bulk copying of index inputs should be in codec as well. Currently its not only wrong as you noted, but makes assumptions about the implementation. But i didn't want to just shove this into CodecProvider since it doesnt really belong there. Finally, I do think the CodecProvider has a place after we fix these names. But I dont think it should be really any more than name -> Codec resolution...

currently it does too much. But to fix this, we really want to remove all the special per-field map, etc stuff it has... and this means factoring PerFieldCodecWrapper back out into codecs (in my opinion this should be PerFieldPostingsFormat, and just an 'ordinary' Codec). And for that to work correctly, we need FieldInfos reading/writing under codec control so that this per-field stuff can be private to PerFieldPostingsFormat.... So there is a ton to do, although I made a branch I'm kinda concerned about doing a bunch of renaming and keeping things in sync... maybe I should ignore this though. But for now I've been trying to figure out any way we can do this in individual incremental steps/issues directly on trunk, its always nice to make progress that way.

16. IMHO you could merge the branch back into trunk, we are allowed to experiment and the current patch already improves the API and shows what to do next.
17. bq. IMHO you could merge the branch back into trunk, +1 This baby step (rote move of stored fields under codec control) is great.
18. **body:** Well technically its Codec*Provider* control, which is global, until we fix the codec hierarchy. But you can also subclass segmentinfosreader/writer in the same expert way (even harder/impossible to "fix"), so how about for now i just mark these methods "expert", merge, and followup with an issue to fix the codec hierarchy? (and later we need to make the apis more general and all these other things)
    label: code-design
19. I opened LUCENE-3490 to try to restructure our current "Codec" idea so that we can put more of the index underneath it.
20. **body:** OK I think the stored fields are all cleaned up in the lucene2621 branch: the issues Andrzej encountered when trying to extend it should no longer be an issue: fdx/fdt are no longer in IndexFileNames (but codec-private), codecs can override how merging happens for stored fields (the default is just the obvious impl), the indexinput/output "raw" methods for bulk merging are removed from the abstract API and only inside DefaultFieldsReader/Writer, DefaultFieldsWriter overrides merge() and uses bulk optimisations when the paired-up reader is also a DefaultFieldsReader Before merging to trunk, i think we need a SimpleText stored fields implementation to ensure everything is ok... I'll try to do this next.
    label: code-design
21. All tests pass with the new simpletext stored fields impl: I'd like to merge the branch to trunk. attached is a patch between trunk and branch.
22. +1 looks good (as far as I can tell!), even SegmentMerger starts making sense now :)
23. +1! Very exciting to finally have stored fields under codec control...
24. **body:** Attached is a new patch between trunk and branch. I think its at a point ready for merging. * term vectors and fieldinfos move to codec. * segmentinfos is moved to codec (before you could only realistically tweak a few things). * term vectors are cut over to flex apis * much better testing of term vectors in checkindex. * added simpletext impls of term vectors, fieldinfos, and segmentinfos. After this I would propose closing this issue and opening followup issues for: * make a new more efficient term vector implementation for 4.0, the existing one would go to preflex, and preflex impl should reorder the terms correctly to UTF8 order (this is a bug all along in trunk, not caused here!) * see if we can remove the global .fnx file completely, as its not per-segment and i'm not sure its totally necessary, perhaps the field number consistency can be achieved with another mechanism. Otherwise, we should add a codec hack/hook at least so that preflexRW can write segments without .fnx files. * make preflex implementations of the other various reader/writers so that our 4.0 impls are clean and don't contain backwards compatibility code, and so that we have more realistic testing of backwards with PreFlexRW. * allow adding offsets to the postings lists impls either startOffset/endOffset() or via attribute like term vectors do in this patch, so that a D&Penum can retrieve the offsets at a position. this could make highlighting much faster without having to use vectors. * try to make a few other things like deletes extendable via codec * figure out a good design to cut over norms to DocValues. * add a SimpleTextDocValues, its sorely needed.
    label: code-design
25. Nice!