Item 329
git_comments:

1. nocommit writeVInt instead?
2. nocommit writeVInt instead?

git_commits:

1. **summary:** LUCENE-3892: add nocommits
   message: LUCENE-3892: add nocommits git-svn-id: https://svn.apache.org/repos/asf/lucene/dev/branches/pforcodec_3892@1362015 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Add a useful intblock postings format (eg, FOR, PFOR, PFORDelta, Simple9/16/64, etc.)
   description: On the flex branch we explored a number of possible intblock encodings, but for whatever reason never brought them to completion. There are still a number of issues opened with patches in different states. Initial results (based on prototype) were excellent (see http://blog.mikemccandless.com/2010/08/lucene-performance-with-pfordelta-codec.html). I think this would make a good GSoC project.

jira_issues_comments:

1. Hi, I have submitted my proposal. Comments are welcome! Also, I made it public: http://www.google-melange.com/gsoc/proposal/review/google/gsoc2012/billybob/1
2. That's great Han, I'll have a look. I can be a mentor for this...
3. The proposal at http://www.google-melange.com/gsoc/proposal/review/google/gsoc2012/billybob/1 looks great! Some initial feedback: * There are actually more than 2 codecs (eg we also have Lucene3x, SimpleText, sep/intblock (abstract), random codecs/postings formats for testing...), but our default codec now is Lucene40. * I think you can use the existing abstract sep/intblock classes (ie, they implement layers like FieldsProducer/Consumer...), and then you can "just" implement the required methods (eg to encode/decode one int[] block). * We may need to tune the skipper settings, based on profiling results from skip-intensive (Phrase, And) queries... since it's currently geared towards single-doc-at-once encoding. I don't think we should try to make a new skipper impl here... (there is a separate issue for that). * Maybe explore the combination of pulsing and PForDelta codecs; seems like the combination of those two could be important, since for low docFreq terms, retrieving the docs is now more expensive...
4. Hi Mike, I have changed my proposal a bit, but here are some questions here: {quote} * There are actually more than 2 codecs (eg we also have Lucene3x, SimpleText, sep/intblock (abstract), random codecs/postings formats for testing...), but our default codec now is Lucene40. {quote} Yes, but it seems that our baseline will be Lucene40 and Pulsing? Lucene3x is read-only, and other approaches are not productive. And, what is random codec? Does it mean to randomly pick up a codec for user? {quote} * I think you can use the existing abstract sep/intblock classes (ie, they implement layers like FieldsProducer/Consumer...), and then you can "just" implement the required methods (eg to encode/decode one int[] block). {quote} And this was my initial thought about the PForDelta interface: The class hierarchy will be as below (quite similar to pulsing): * PForDeltaPostingsFormat(extends PostingsFormat): It will define global behaviors such as file suffix, and provide customized FieldsWriter/Reader * PForDeltaFieldsWriter(extends FieldsConsumer): It will define how terms,docids,freq,offset are written into posting files. inner classes include: ** PForDeltaTermsConsumer(extends TermsConsumer) ** PForDeltaPostingsConsumer(extends

PostingsConsumer) * PForDeltaFieldsReader(extends FieldsProducer): It will define how postings are read from index, and provide *Enum class to iterate docids, freqs etc. inner classes include: ** PForDeltaFieldsEnum(extends FieldsEnum) ** PForDeltaTermsEnum(extends TermsEnum) ** PForDeltaDocsEnum(extends DocsEnum) ** PForDeltaDocsAndPositonsEnum(extends DocsAndPostionsEnum) ** PForDeltaTerms(extends Terms) It seems that "BlockTermsReader/Writer" have already implement those subclasses, and we can just pass our Postings(Writer/Reader)Base as an argument, like PatchedFrameOfRefCodec::fieldsConsumer() does. Then, to introduce PForDeltaCodec into trunk, we should also introduce the "fixed codec"? Also, why isn't lucene40codec implemented with this line? {quote} * We may need to tune the skipper settings, based on profiling results from skip-intensive (Phrase, And) queries... since it's currently geared towards single-doc-at-once encoding. I don't think we should try to make a new skipper impl here... (there is a separate issue for that). {quote} It seems that skip settings are not so related to backend codec? Do you mean the nocommit line in FixedPostingsWriterImpl.java:117 ? {quote} * Maybe explore the combination of pulsing and PForDelta codecs; seems like the combination of those two could be important, since for low docFreq terms, retrieving the docs is now more expensive... {quote} Yes, it seems that if PForDelta outperforms current approaches, a Pulsing version will work better. This feature will also come as "phase 2".

5. Thank all of you for providing me this opportunity! Let us begin!
6. Hi Billy, I'm very excited your proposal is accepted! Congrats :) Now the fun work begins...
7. It's quite strange that sometimes I cannot access repo1.maven.org, therefore "ant ivy-boostrap" & "ant resolve" will fail to work.(Since I'm in China, the network connection might be limited). Once Mike and I hoped to make things work by configuring "lucene/common-build.xml" & "dev-tools/scripts/poll-mirrors.pl" to another maven mirror, listed in http://docs.codehaus.org/display/MAVENUSER/Mirrors+Repositories. Unfortunately, the main site "repo1.maven.org" was configured into ivy-2.2.0.jar, and even we pass "ant ivy-bootstrap", "ant resolve" still fails. Well, here is how I get things work(too ugly, hope a better suggestion!): change /etc/hosts, and redirect current maven site to a mirror with same directory structure, for example: 194.8.197.22 repo1.maven.org # to http://mirror.netcologne.de/
8. Phew, I'm glad to hear you got it working! So "ant resolve" finished successfully?
9. Yes, and "ant test" is running now. Maybe we can configure something to avoid the ugly hack?
10. Maybe a good solution is if we have an ant property (that we somehow pass to ivy), and we conditionally set it in ant by default to a server we know that works in china, if the "${user.language}"="zh" ?
11. Thank you, Robert! But currently, the maven mirror in China(http://mirrors.redv.com/maven2) is not available. And can we pass a property to ivy to replace the "repo1*" stuff?
12. can you remove your hack and try this patch?
13. Thank you Robert! The patch works well.
14. Patch does not yet fix ivy-bootstrap. Ivy-bootstrap still only tries repo1.maven.org. We need a different strategy for that: either we depend on try-catch from ant contrib (undesired), use custom ant task (grrrr), or use a chain of targets with fail-on-error=false unless the file already exists and checksum at the end... Lemme see if i can fix ivy-bootstrap, too!
15. updated patch with also logic for ivy-bootstrap. if repo1.maven.org fails, we try the same china-friendly mirror (currently http://mirror.netcologne.de/maven2). We disable fail-on-error, instead sha1-checksum the result at the end to determine real success or not (and if it fails that, prints a message suggesting you manually download it)
16. I will commit this patch: please let us know if you have more problems from china! :)
17. OK, and thanks for the new commit!
18. A postings format named VSEncoding also seems promising! It is available here: http://integerencoding.isti.cnr.it/ And license compatible: https://github.com/maropu/integer_encoding_library/blob/master/LICENSE
19. Here is a initial implementation of PForPostingsFormat. It is registered in oal.codecs.mockrandom.MockRandomPostingsFormat, and all tests have passed (Maybe I should modify some other mock files as well?). This version is orginally inspired by the pfor and pfor2 impls in bulk_branch, mostly by the idea of pfor. Currently, the compressed data consists of three parts: header, normal area, and excpetion area. The normal area encodes each small value as b bits, as well as exception values. The exception area stores each large value directly, possibly as 8,16,or 32 bits. NumFrameBits range from 1-32 are all supported. I haven't test the performance, but there are some known bottlenecks: For example, data = {0, 0xffffffff, 0, 1, 0, 1, 0}, numFrameBits=1, then the following '1's will be forced as exceptions, which will dramatically increase compressed size.
20. Awesome progress! Nice to have a dirt path online that we can then iterate from ... Hmm, I'm seeing some test failures when I run: {noformat} ant test -Dtests.postingsformat=PFor {noformat} Eg,

TestNRTThreads, TestShardSearching, TestTimeLimitingCollector. Remember to add the standard copyright headers to each new source file... We don't have to do this now, but I wonder if we can share code w/ the packed ints impl we have, instead generating another one with the .py source. TestDemo makes a nice TestMin... I usually start with TestDemo when testing scary new code, and then it's a huge milestone once TestDemo passes :) We should definitely cutover to BlockTree terms dict (I would upgrade that TODO to a nocommit!). I suspect that wrapping the blocks byte[] as ByteBuffer and then IntBuffer is going to be too costly per decode so we should init them once and re-use (upgrade that TODO to a nocommit).

21. Ah, yes, I forgot to use -Dtests.postingsformat...I can see the errors now. {quote} TestDemo makes a nice TestMin... I usually start with TestDemo when testing scary new code, and then it's a huge milestone once TestDemo passes {quote} Hmm, that means I should remove TestMin.java? This testcase works fine for the patch. {quote} We should definitely cutover to BlockTree terms dict (I would upgrade that TODO to a nocommit!). {quote} I'm not quite familiar with these sign stuff, shall I change all the "TODO" sign into "nocommit"? Are the signs related to documentation, or just marked to remember not to commit current codes?
22. bq. Hmm, that means I should remove TestMin.java? This testcase works fine for the patch. Oh it's fine to keep TestMin now that you wrote it ... I was just saying that TestDemo is the test I run when I want the most trivial test for a new big change. {quote} I'm not quite familiar with these sign stuff, shall I change all the "TODO" sign into "nocommit"? Are the signs related to documentation, or just marked to remember not to commit current codes? {quote} Sorry - this is just a convention I use: I put a // nocommit comment whenever there's a "blocker" to committing; this way I can grep for nocommit to see what still needs fixing... and towards the end, nocommits will often be downgraded to TODOs since on closer inspection they really don't have to block committing...
23. Ah, just cannot wait for a performance optimization! This version should now pass all tests below: ant test-core -Dtests.postingsformat=PFor It fixes: 1) trailing forced exceptions will be ignored and encoded as normal value; 2) IntBuffer is maintained at IndexInput/Output level; 3) Former nocommit issues such as BlockTreeTerms* and code licence. The patch also contains a minimal change with the help of Robert's patch: https://issues.apache.org/jira/secure/attachment/12530685/LUCENE-4102.patch. Hope Dawid will commit the complete version into trunk soon! I'll try to optimize these codes later.
24. Excellent! All tests also pass for me w/ PFor postings format as well... this is a great starting point :) One Solr test failed (ContentStreamTest)... but I think it was false failure... I did notice the tests seem to run slower, especially certain ones eg TestJoinUtil. Still missing a couple license headers (TestMin, TestCompress)... I ran a quick perf test using http://code.google.com/a/apache-extras.org/p/luceneutil on a 10M doc Wikipedia index. Indexing time is ~18% slower than Lucene40PostingsFormat (1071 sec vs 1261 sec). But more important is the slower search times: {noformat} Task QPS base StdDev base QPS pfor StdDev pfor Pct diff Phrase 8.52 0.50 4.43 0.40 -55% - -39% SloppyPhrase 12.52 0.39 7.87 0.51 -43% - -30% AndHighMed 67.69 2.82 44.22 1.47 -39% - -29% SpanNear 5.19 0.12 3.90 0.28 -31% - -17% PKLookup 112.16 1.71 95.61 1.30 -17% - -12% AndHighHigh 13.22 0.34 11.86 0.72 -17% - -2% Wildcard 46.04 0.37 41.68 4.45 -19% - 1% Fuzzy1 50.11 2.03 48.06 1.91 -11% - 3% OrHighMed 9.26 0.48 8.90 0.37 -12% - 5% OrHighHigh 12.28 0.56 11.83 0.49 -11% - 5% TermBGroup1M1P 40.47 1.94 39.88 2.51 -11% - 10% Fuzzy2 53.71 2.66 53.01 2.08 -9% - 7% TermGroup1M 36.46 1.21 35.99 1.58 -8% - 6% TermBGroup1M 55.53 1.99 55.26 2.68 -8% - 8% Respell 69.71 4.49 69.73 2.07 -8% - 10% Term 94.38 7.62 94.96 12.19 -18% - 23% Prefix3 41.63 0.34 42.21 5.82 -13% - 16% IntNRQ 7.08 0.15 7.28 1.29 -17% - 23% {noformat} The queries that do skipping are quite a bit slower; this makes sense, since on skip we do a full block decode. A smaller block size (we use 128 now right?) should help I think. It's strange that the non-skipping queries (Term, OrHighMed, OrHighHigh) don't show any performance gain ... maybe we need to optimize the decode... or it could be the removal of the bulk api is hurting us here. I'm also curious if we tried a pure FOR (no patching, so we must set numBits according to the max value = larger index but hopefully faster decode) if the results would improve...
25. Thanks Mike, we have so much details to help optimize! bq.Still missing a couple license headers (TestMin, TestCompress)... Ok, I'll add them later. bq.I ran a quick perf test using http://code.google.com/a/apache-extras.org/p/luceneutil on a 10M doc Wikipedia index. The script is wonderful! But the wiki data is missing? Can I get it from a wiki dump instead? bq.Indexing time is ~18% slower than Lucene40PostingsFormat (1071 sec vs 1261 sec). Yes, it is expected, actually it scans every block 33 times to estimate metadata such as numFrameBits and numExceptions.
26. Hi Billy, bq. Can I get it from a wiki dump instead? You can download it at http://people.apache.org/~mikemccand/enwiki-20120502-lines-1k.txt.lzma That's ~6.3 GB (compressed) and 28.7 GB (decompressed); it's the 2012/05/02 Wikipedia en export, filtered to plain text and then broken into 33.3 M ~1 KB sized docs. I can help you get the luceneutil env set up... {quote} bq. Indexing

time is ~18% slower than Lucene40PostingsFormat (1071 sec vs 1261 sec). Yes, it is expected, actually it scans every block 33 times to estimate metadata such as numFrameBits and numExceptions. {quote} OK, in that case I'm surprised it's only ~18% slower!

27. OK, here is a result I tried to reproduce with Mike's test script: Indexing time: trunk: 2396 sec patch: 2793 sec Searching time: {noformat} TaskQPS Lucene40StdDev Lucene40 QPS PFor StdDev PFor Pct diff AndHighMed 22.76 0.54 14.68 1.00 -41% - -29% SloppyPhrase 3.58 0.17 2.46 0.27 -41% - -19% SpanNear 5.90 0.09 4.08 0.37 -38% - -23% AndHighHigh 10.00 0.17 8.08 0.57 -26% - -11% Phrase 1.68 0.07 1.45 0.17 -27% - 0% Respell 37.65 0.74 33.41 1.04 -15% - -6% Fuzzy1 38.00 1.60 34.37 1.06 -15% - -2% IntNRQ 4.27 0.33 3.87 0.19 -19% - 3% Fuzzy2 16.35 0.60 15.02 0.31 -13% - -2% Wildcard 30.24 0.57 28.24 1.85 -14% - 1% PKLookup 85.82 5.04 83.25 2.81 -11% - 6% Prefix3 19.20 0.40 19.19 1.46 -9% - 9% OrHighMed 9.25 0.59 9.41 0.70 -11% - 16% TermGroup1M 11.46 0.62 11.74 0.81 -9% - 15% OrHighHigh 3.15 0.17 3.28 0.23 -8% - 17% TermBGroup1M1P 19.28 0.38 20.32 1.14 -2% - 13% TermBGroup1M 6.23 0.21 6.71 0.46 -3% - 19% Term 30.86 1.52 34.34 3.26 -4% - 28% {noformat} It is done on a 64bit AMD server with Java 1.7.0.
28. The new "3892_pfor" patch fixed some "SuppressingCodec" stuff since last two weeks. And the "3892_for" lazily implements "For" postingsformat based on current codes. These two patches are temporary separated, in order to prevent performance reduction for the sake of method overriding. Currently, blocksize ranges from 32 to 128 are tested on both two patches. However, for those skipping-intensive queries, there is no significant performance gain when smaller blocksize was applied. Here is a previous result for PFor, with blockSize=64, comparing with 128(in brackets): {noformat} Task QPS Base StdDev Base QPS PFor StdDev PFor Pct diff Phrase 4.93 0.36 3.10 0.33 -47% - -25% (-47% - -25%) AndHighMed 27.92 2.26 19.16 1.72 -42% - -18% (-37% - -15%) SpanNear 2.73 0.16 1.96 0.24 -40% - -14% (-36% - -13%) SloppyPhrase 4.19 0.21 3.20 0.30 -34% - -12% (-30% - -6%) Wildcard 19.44 0.87 17.11 0.94 -20% - -2% (-17% - 3%) AndHighHigh 7.50 0.38 6.61 0.59 -23% - 1% (-19% - 6%) IntNRQ 4.06 0.52 3.88 0.35 -22% - 19% (-16% - 24%) Prefix3 31.00 1.69 30.45 2.29 -13% - 11% (-6% - 20%) OrHighHigh 4.16 0.47 4.11 0.34 -18% - 20% (-14% - 27%) OrHighMed 4.98 0.59 4.94 0.41 -18% - 22% (-14% - 27%) Respell 40.29 2.11 40.11 2.13 -10% - 10% (-15% - 2%) TermBGroup1M 20.50 0.32 20.52 0.80 -5% - 5% (1% - 10%) TermGroup1M 13.51 0.43 13.61 0.40 -5% - 7% (1% - 9%) Fuzzy1 43.20 1.83 44.02 1.95 -6% - 11% (-11% - 1%) PKLookup 87.16 1.78 89.52 0.94 0% - 5% (-2% - 7%) Fuzzy2 16.09 0.80 16.54 0.77 -6% - 13% (-11% - 6%) Term 43.56 1.53 45.26 3.84 -8% - 16% (2% - 26%) TermBGroup1M1P 21.33 0.64 22.24 1.23 -4% - 13% (0% - 14%) {noformat} Also, the For postingsformat shows few performance change. So I suppose the bottleneck isn't in this method: PForUtil.patchException. Here is an example with blockSize=64: {noformat} Task QPS Base StdDev Base QPS For StdDev For Pct diff Phrase 5.03 0.45 3.30 0.43 -47% - -18% AndHighMed 28.05 2.33 18.83 1.77 -43% - -19% SpanNear 2.69 0.18 1.94 0.25 -40% - -12% SloppyPhrase 4.19 0.20 3.22 0.35 -34% - -10% AndHighHigh 7.61 0.46 6.41 0.54 -27% - -2% Respell 41.36 1.65 37.94 2.42 -17% - 1% Wildcard 19.20 0.77 17.89 0.99 -15% - 2% OrHighHigh 4.22 0.37 3.94 0.32 -21% - 10% OrHighMed 5.06 0.46 4.73 0.39 -21% - 11% Fuzzy1 44.15 1.31 42.38 1.74 -10% - 2% Fuzzy2 16.48 0.59 15.84 0.76 -11% - 4% TermGroup1M 13.32 0.35 13.44 0.53 -5% - 7% PKLookup 87.70 1.81 88.62 1.22 -2% - 4% TermBGroup1M 20.14 0.47 20.40 0.59 -3% - 6% Prefix3 30.31 1.49 31.08 2.26 -9% - 15% TermBGroup1M1P 21.13 0.46 21.79 1.42 -5% - 12% IntNRQ 3.96 0.45 4.14 0.46 -16% - 31% Term 43.07 1.51 46.06 4.50 -6% - 21% {noformat}
29. There's a potential bottleneck during method calling...Here is an example for PFor, with blocksize=128, exception rate = 97%, normal value <= 2 bits, exception value <= 32 bits: {noformat} Decoding normal values: 4703 ns Patching exceptions: 5797 ns Single call of PForUtil.decompress totally takes: 58318 ns {noformat} In addition, it costs about 4000ns to record the time span.
30. On the For patch ... we shouldn't encode/decode numInts right? It's always 128? Up above, in ForFactory, when we readInt() to get numBytes ... it seems like we could stuff the header numBits into that same int and save checking that in FORUtil.decompress.... I think there are a few possible ideas to explore to get faster PFor/For performance: * Get more direct access to the file as an int[]; eg MMapDir could expose an IntBuffer from its ByteBuffer (saving the initial copy into byte[] that we now do). Or maybe we add IndexInput.readInts(int[]) and dir impl can optimize how that's done (MMapDir could use Unsafe.copyBytes... except for little endian architectures ... we'd probably have to have separate specialized decoder rather than letting Int/ByteBuffer do the byte swapping). This would require the whole file stays aligned w/ int (eg the header must be 0 mod 4). * Copy/share how oal.packed works, i.e. being able to waste a bit to have faster decode (eg storing the 7 bit case as byte[], wasting 1 bit for each value). * Skipping: can we partially decode a block? EG if we are skipping and we know we only want values after the 80th one, then we shouldn't decode those first 80... * Since doc/freq are "aligned", when we store pointers to a given spot, eg in the terms dict or in skip data, we should only store the offset once

(today we store it twice). * Alternatively, maybe we should only save skip data on doc/freq block boundaries (prox would still need skip-within-block). * Maybe we should store doc & frq blocks interleaved in a single file (since they are "aligned") and then skip would skip to the start of a doc/frq block pair. Other ideas...?

31. Oh, thank you Mike! I haven't thought too much about those skipping policies. bq. Up above, in ForFactory, when we readInt() to get numBytes ... it seems like we could stuff the header numBits into that same int and save checking that in FORUtil.decompress.... Ah, yes, I just forgot to remove the redundant codes. Here is a initial try to remove header and call ForDecompressImpl directly in readBlock():with For, blockSize=128. Data in bracket show prior benchmark. {noformat} Task QPS Base StdDev Base QPS For StdDev For Pct diff Phrase 4.99 0.37 3.57 0.26 -38% - -17% (-44% - -18%) AndHighMed 28.91 2.17 22.66 0.82 -29% - -12% (-38% - -9%) SpanNear 2.72 0.14 2.22 0.13 -26% - -8% (-36% - -8%) SloppyPhrase 4.24 0.26 3.70 0.16 -21% - -3% (-33% - -6%) Respell 40.71 2.59 37.66 1.36 -16% - 2% (-18% - 0%) Fuzzy1 43.22 2.01 40.66 0.32 -10% - 0% (-12% - 0%) Fuzzy2 16.25 0.90 15.64 0.26 -10% - 3% (-12% - 3%) Wildcard 19.07 0.86 19.07 0.73 -8% - 8% (-21% - 3%) AndHighHigh 7.76 0.47 7.77 0.15 -7% - 8% (-21% - 10%) PKLookup 87.50 4.56 88.51 1.24 -5% - 8% (-2% - 5%) TermBGroup1M 20.42 0.87 21.32 0.74 -3% - 12% (2% - 10%) OrHighMed 5.33 0.68 5.61 0.14 -9% - 23% (-16% - 25%) OrHighHigh 4.43 0.53 4.69 0.12 -8% - 23% (-15% - 24%) TermGroup1M 13.30 0.34 14.31 0.40 2% - 13% (0% - 13%) TermBGroup1M1P 20.92 0.59 23.71 0.86 6% - 20% (-1% - 22%) Prefix3 30.30 1.41 35.14 1.76 5% - 27% (-14% - 21%) IntNRQ 3.90 0.54 4.58 0.47 -7% - 50% (-25% - 33%) Term 42.17 1.55 52.33 2.57 13% - 35% (1% - 33%) {noformat} -The improvement is quite general. However, I still suppose this just benefits from less method calling. I'm trying to change the PFor codes, and remove those nested call.- (this is not actually true, since I was using percentage diff instead of QPS during comparison) bq. Get more direct access to the file as an int[]; ... Ok, this will be considered when the pfor+pulsing is completed. I'm just curious why we don't have readInts in ora.util yet... bq. Skipping: can we partially decode a block? ... The pfor-opt approach(encode lower bits of exception in normal area, and other bits in exception area) natually fits "partially decode a block", that'll be possible when we optimize skipping queries.
32. And result for PFor(blocksize=128): {noformat} Task QPS Base StdDev Base QPS PFor StdDev PFor Pct diff Phrase 4.87 0.36 3.39 0.18 -38% - -20% (-47% - -25%) AndHighMed 27.78 2.35 21.13 0.52 -31% - -14% (-37% - -15%) SpanNear 2.70 0.14 2.20 0.11 -26% - -9% (-36% - -13%) SloppyPhrase 4.17 0.15 3.77 0.21 -17% - 0% (-30% - -6%) Respell 39.97 1.56 37.65 1.95 -14% - 3% (-15% - 2%) Wildcard 19.08 0.77 18.33 0.92 -12% - 5% (-17% - 3%) Fuzzy1 42.29 1.13 40.78 1.44 -9% - 2% (-11% - 1%) AndHighHigh 7.61 0.55 7.45 0.08 -9% - 6% (-19% - 6%) Fuzzy2 15.79 0.55 15.64 0.70 -8% - 7% (-11% - 6%) PKLookup 86.71 2.13 88.92 2.24 -2% - 7% (-2% - 7%) TermGroup1M 13.04 0.23 14.03 0.40 2% - 12% (1% - 9%) IntNRQ 3.97 0.48 4.35 0.61 -15% - 41% (-16% - 24%) TermBGroup1M1P 21.04 0.35 23.20 0.60 5% - 14% (0% - 14%) TermBGroup1M 19.27 0.47 21.28 0.84 3% - 17% (1% - 10%) OrHighHigh 4.13 0.47 4.63 0.27 -5% - 34% (-14% - 27%) OrHighMed 4.95 0.59 5.58 0.34 -5% - 35% (-14% - 27%) Prefix3 30.33 1.36 34.26 2.14 1% - 25% (-6% - 20%) Term 41.99 1.19 50.75 1.72 13% - 28% (2% - 26%) {noformat} -It works, and it is quite interesting that StdDev for Term query is reduced significantly.- (same as last comment, when comparing two versions directly(method call vs. unfolded, the improvement is somewhat noisy))
33. The For index is 5.2 GB vs 4.9 GB for vInt: not bad to have only 5% increase in index size when using For PF (10M wikipedia index). {quote} Get more direct access to the file as an int[]; eg MMapDir could expose an IntBuffer from its ByteBuffer (saving the initial copy into byte[] that we now do). {quote} I tested this, by making hacked up changes to Billy's For patch requiring MMapDirectory and pulling an IntBuffer directly from its ByteBuffer, saving one copy of bytes into the byte[] first. But, curiously, it didn't seem to improve things much: {noformat} Task QPS base StdDev base QPS for StdDev for Pct diff AndHighMed 24.32 0.60 14.24 0.41 -44% - -38% PKLookup 131.98 3.09 108.35 1.47 -20% - -14% AndHighHigh 5.36 0.18 4.66 0.02 -16% - -9% Phrase 1.48 0.02 1.33 0.10 -18% - -2% SloppyPhrase 1.40 0.04 1.26 0.03 -13% - -5% SpanNear 1.14 0.01 1.04 0.02 -10% - -6% IntNRQ 12.13 0.70 11.27 0.46 -15% - 2% Prefix3 34.51 1.17 34.11 1.28 -8% - 6% Fuzzy1 90.63 1.74 89.68 1.46 -4% - 2% Respell 77.22 2.62 76.99 1.62 -5% - 5% Wildcard 11.84 0.40 12.20 0.37 -3% - 9% Fuzzy2 34.34 0.82 36.16 1.08 0% - 11% TermBGroup1M1P 4.71 0.11 5.02 0.18 0% - 12% OrHighMed 7.87 0.28 8.50 0.55 -2% - 19% TermBGroup1M 3.47 0.03 3.78 0.03 7% - 11% TermGroup1M 2.96 0.01 3.25 0.03 8% - 11% OrHighHigh 3.55 0.12 3.91 0.21 0% - 20% Term 9.72 0.28 10.87 0.44 4% - 19% {noformat} Maybe, instead, reading into an int[] and decoding from an int array (hopefully avoiding bounds checks) will be faster than calling IntBuffer.get for each encoded int...
34. The *unfold_method.patch just remove the nested call of PForDecompressImpl.decode, and also clip out numBytes information for ForPF.

35. OK I created a branch and committed last For patch: https://svn.apache.org/repos/asf/lucene/dev/branches/pforcodec_3892
36. OK, just reproduce your test. But Mike, are we using a same task file? Our relative speeds for different queries are not the same. {noformat} Task QPS Base StdDev Base QPS For StdDev For Pct diff Phrase 5.07 0.45 3.76 0.19 -35% - -14% (-44% - -18%) AndHighMed 28.32 2.34 22.67 0.67 -28% - -10% (-38% - -9%) SpanNear 2.72 0.13 2.36 0.14 -22% - -3% (-36% - -8%) SloppyPhrase 4.18 0.20 3.83 0.15 -16% - 0% (-33% - -6%) Respell 42.02 1.83 38.86 2.30 -16% - 2% (-18% - 0%) Fuzzy1 44.96 1.58 42.85 1.69 -11% - 2% (-12% - 0%) Fuzzy2 16.78 0.69 16.34 0.68 -10% - 5% (-12% - 3%) PKLookup 89.11 2.15 87.33 2.19 -6% - 2% (-2% - 5%) AndHighHigh 7.61 0.44 7.69 0.21 -7% - 10% (-21% - 10%) Wildcard 19.50 0.91 20.02 0.72 -5% - 11% (-21% - 3%) TermBGroup1M 20.82 0.37 21.73 0.69 0% - 9% (2% - 10%) TermGroup1M 13.79 0.13 14.61 0.32 2% - 9% (1% - 9%) IntNRQ 4.11 0.56 4.56 0.56 -14% - 43% (-25% - 33%) TermBGroup1M1P 21.45 0.75 24.00 0.51 5% - 18% (-1% - 22%) OrHighMed 5.08 0.49 5.73 0.15 0% - 28% (-16% - 25%) OrHighHigh 4.22 0.39 4.78 0.13 1% - 28% (-15% - 24%) Prefix3 30.91 1.63 35.65 2.02 3% - 28% (-14% - 21%) Term 44.36 1.87 54.01 1.96 12% - 31% (-1% - 33%) {noformat}
37. bq. But Mike, are we using a same task file? Our relative speeds for different queries are not the same. Sorry, I'm using a hand edited "hard" tasks file; I'll commit & push to luceneutil. But, separately: each run picks a different subset of the tasks from each category to run, so results from one run to another in general aren't comparable unless we fix the random seed it uses.
38. For decompressing phase, replace the use of IntBuffer with a direct int[] to int[] decoder. Method convert() is supposed to be performant enough...coz it is not different from the inner implementation of IntBuffer.get(), i.e.http://massapi.com/source/jdk1.6.0_17/src/java/nio/Bits.java.html, line 193. However, result isn't interesting. Hmm, there is an extra block of memory write here, which Mike wanted to avoid in previous patch. That should be the cause.
39. Now remove the memory write codes, and replace IntBuffer.get() with getInt(byte,byte,byte,byte), since this patch contains method unfolding, there is no actually difference...Seems that we're paying attenting on a wrong point. {noformat} Task QPS Base StdDev Base QPS For StdDev For Pct diff Phrase 5.02 0.46 3.66 0.30 -38% - -13% (-38% - -17%) AndHighMed 28.08 2.29 23.04 1.01 -27% - -6% (-29% - -12%) SpanNear 2.69 0.16 2.30 0.19 -25% - 0% (-26% - -8%) SloppyPhrase 4.18 0.22 3.83 0.18 -16% - 1% (-21% - -3%) Respell 41.92 2.15 39.54 2.45 -15% - 5% (-16% - 2%) Fuzzy1 44.47 1.99 43.34 3.07 -13% - 9% (-10% - 0%) Wildcard 19.70 1.06 19.60 1.16 -11% - 11% (-8% - 8%) Fuzzy2 16.54 0.86 16.52 1.16 -11% - 12% (-10% - 3%) PKLookup 87.32 2.47 88.62 1.33 -2% - 6% (-5% - 8%) AndHighHigh 7.55 0.43 7.84 0.15 -3% - 12% (-7% - 8%) TermBGroup1M 19.86 0.14 21.41 0.70 3% - 12% (-3% - 12%) TermGroup1M 13.35 0.17 14.40 0.38 3% - 12% (2% - 13%) IntNRQ 4.10 0.57 4.45 0.73 -20% - 46% (-7% - 50%) TermBGroup1M1P 21.29 0.63 23.45 0.82 3% - 17% (6% - 20%) Prefix3 31.13 1.71 35.53 2.90 0% - 30% (5% - 27%) OrHighMed 4.96 0.61 5.83 0.35 -1% - 42% (-9% - 23%) OrHighHigh 4.13 0.49 4.87 0.29 0% - 41% (-8% - 23%) Term 42.93 1.17 52.11 2.21 13% - 30% (13% - 35%) {noformat} It is compared with result in https://issues.apache.org/jira/browse/LUCENE-3892?focusedCommentId=13396987&page=com.atlassian.jira.plugin.system.issuetabpanels:comment-tabpanel#comment-13396987
40. And same codes with the wikimediumhard.tasks file.(This is really a hard testcase, since QPS are so small that we can hardly depend on Pct Diff :)) {noformat} Task QPS Base StdDev Base QPS For StdDev For Pct diff AndHighMed 10.76 0.21 6.47 0.32 -43% - -35% AndHighHigh 2.89 0.08 2.57 0.19 -20% - -1% SpanNear 0.60 0.01 0.55 0.01 -11% - -6% SloppyPhrase 0.61 0.01 0.57 0.01 -9% - -3% PKLookup 87.72 2.61 86.28 1.48 -6% - 3% Fuzzy1 36.22 1.14 35.90 0.97 -6% - 5% Phrase 1.22 0.03 1.22 0.08 -9% - 8% Respell 32.84 0.92 33.55 0.87 -3% - 7% IntNRQ 3.66 0.35 3.74 0.08 -8% - 15% Fuzzy2 21.62 0.66 22.10 0.51 -3% - 7% Prefix3 13.30 0.49 14.09 0.76 -3% - 15% OrHighMed 3.43 0.16 3.65 0.45 -10% - 25% OrHighHigh 1.66 0.09 1.79 0.22 -10% - 28% Wildcard 3.39 0.14 3.74 0.20 0% - 21% TermBGroup1M1P 1.84 0.03 2.10 0.16 3% - 25% TermGroup1M 1.14 0.03 1.34 0.10 5% - 29% TermBGroup1M 1.49 0.05 1.78 0.13 7% - 32% Term 3.49 0.13 4.38 0.65 2% - 49% {noformat}
41. I was curious how much the "layers" (SepPostingsReader, FixedIntBlock.IntIndexInput, ForFactor) between the FOR block decode and the query scoring were hurting performance, so I wrote a specialized scorer (BlockTermScorer) for just TermQuery. The scorer is only used if the postings format is ForPF, and if no skipping will be done (I didn't implement advance...). The scorer reaches down and holds on to the decoded int[] buffer, and then does its own adding up of the doc deltas, reading the next block, etc. The baseline is the current branch (not trunk!): {noformat} Task QPS base StdDev base QPS patch StdDev patch Pct diff Wildcard 10.31 0.40 10.10 0.17 -7% - 3% AndHighHigh 4.90 0.10 4.82 0.15 -6% - 3% Prefix3 28.50 1.06 28.11 0.50 -6% - 4% IntNRQ 9.72 0.46 9.60 0.57 -11% - 9% SloppyPhrase 0.92 0.03 0.92 0.02 -6% - 5% PKLookup 106.21 2.54 105.66 2.07 -4% - 3% Phrase 1.56 0.00 1.56 0.01 -1% - 0%

Fuzzy1 90.33 3.48 90.19 2.25 -6% - 6% Fuzzy2 29.66 0.61 29.64 0.85 -4% - 4% AndHighMed 14.87 0.29 15.02 0.81 -6% - 8% Respell 78.83 2.46 79.62 1.54 -3% - 6% SpanNear 1.18 0.02 1.19 0.04 -4% - 6% TermGroup1M 2.78 0.06 3.28 0.14 10% - 25% OrHighHigh 4.19 0.24 5.04 0.20 9% - 32% OrHighMed 8.21 0.45 9.87 0.23 11% - 30% TermBGroup1M1P 5.11 0.20 6.21 0.26 12% - 31% TermBGroup1M 4.49 0.11 5.49 0.27 13% - 31% Term 8.89 0.58 11.90 1.52 9% - 61% {noformat} Seems like we get a good boost removing the abstractions.

42. It's really interesting the effect of peeling back those abstractions.
43. Yes, really interesting. And that should make sense. -As far as I know, a method with exception handling may be quite slow than a simple if statement check.-(Hmm, now I think this is not true, the improvement should mainly come the framework change) Here is part of the result in my test, with Mike's patch: {noformat} OrHighMed 2.53 0.31 2.57 0.13 -13% - 21% Wildcard 3.86 0.12 3.94 0.38 -10% - 15% OrHighHigh 1.57 0.18 1.61 0.08 -12% - 21% TermBGroup1M1P 1.93 0.03 2.48 0.10 21% - 35% TermGroup1M 1.37 0.02 1.81 0.05 26% - 37% TermBGroup1M 1.17 0.02 1.64 0.07 32% - 47% Term 2.92 0.13 4.46 0.23 38% - 68% {noformat}
44. This version will make PFor work side by side with For. Hmm, the performance results might changed a little, I'll post them later.
45. Thanks Billy, I committed this to the branch.
46. Current branch cannot pass tests like this: {noformat} ant test -Dtestcase=TestConcurrentMergeScheduler -Dtests.method=testFlushExceptions -Dtests.seed=C2ED637AF330E96A -Dtests.postingsformat=For {noformat} Maybe we should handle the IndexOutput more gracefully?
47. And this patch ensures pulsing support for PFor and For postings format.
48. bq. Current branch cannot pass tests like this: Thanks, I committed the patch.
49. Add javadocs for previous codes. I'm still not sure about the IOUtils.closeWhileHandlingException(), I think the exceptions should not be suppressed when out.close() is called? This patch also makes some minor change on randomness in testcase. The Pulsing parts in last patch is not included here, because they doesn't improve performance significantly.
50. bq. I'm still not sure about the IOUtils.closeWhileHandlingException(), I think the exceptions should not be suppressed when out.close() is called? Actually I think you want them to be suppressed, so that the original exception is seen?
51. Docs/cleanup patch looks good, I'll commit to the branch! Thanks.
52. bq. Actually I think you want them to be suppressed, so that the original exception is seen? Not my idea actually, I think the exception should be thrown for out.close()? closeWhileHandlingException() will suppress those exceptions. So in this patch I use out.close() instead of IOUtils.closeWhileHandlingException()
53. bq. Not my idea actually, I think the exception should be thrown for out.close()? closeWhileHandlingException() will suppress those exceptions But the problem is some other exception has already been thrown (because success is false). If out.close then hits a second exception we have to pick which one should be thrown, and I think the original one is better? (Since it's likely the root cause of whatever went wrong).
54. bq. The Pulsing parts in last patch is not included here, because they doesn't improve performance significantly. Here are some tests between For vs PulsingFor, PFor vs PulsingPFor. Run on the 1M docs with wikimediumhard.tasks It is strange that PKLookup still doesn't benefit for FixedBlockInt: {noformat} Task QPS For StdDev ForQPS PulsingForStdDev PulsingFor Pct diff AndHighHigh 23.01 0.33 22.94 0.66 -4% - 4% AndHighMed 56.41 0.76 57.41 1.74 -2% - 6% Fuzzy1 86.74 0.85 82.22 2.39 -8% - -1% Fuzzy2 28.23 0.38 26.15 0.97 -11% - -2% IntNRQ 41.78 1.65 40.78 3.53 -14% - 10% OrHighHigh 14.44 0.34 14.50 0.92 -8% - 9% OrHighMed 30.59 0.77 31.12 1.93 -6% - 10% PKLookup 110.31 2.03 109.22 2.43 -4% - 3% Phrase 8.18 0.44 7.97 0.40 -12% - 8% Prefix3 99.64 2.38 97.09 3.46 -8% - 3% Respell 99.66 0.45 92.76 2.81 -10% - -3% SloppyPhrase 4.28 0.16 4.08 0.13 -11% - 2% SpanNear 4.08 0.13 3.93 0.06 -7% - 0% Term 33.63 1.25 34.06 1.71 -7% - 10% TermBGroup1M 15.54 0.46 15.78 0.56 -4% - 8% TermBGroup1M1P 20.34 0.73 20.62 0.62 -5% - 8% TermGroup1M 19.18 0.52 19.72 0.49 -2% - 8% Wildcard 34.86 0.88 34.27 1.77 -9% - 6% {noformat} {noformat} AndHighHigh 19.98 0.31 19.92 0.26 -3% - 2% AndHighMed 58.21 1.51 57.86 1.18 -5% - 4% Fuzzy1 91.86 1.17 85.86 1.18 -8% - -4% Fuzzy2 32.66 0.58 30.08 0.57 -11% - -4% IntNRQ 33.89 0.82 32.66 1.10 -9% - 2% OrHighHigh 15.79 1.29 14.96 0.67 -16% - 7% OrHighMed 30.31 2.09 28.91 1.67 -15% - 8% PKLookup 112.80 0.81 111.82 2.90 -4% - 2% Phrase 6.14 0.11 6.23 0.10 -1% - 5% Prefix3 147.80 2.88 138.35 2.11 -9% - -3% Respell 118.57 1.18 108.30 1.86 -11% - -6% SloppyPhrase 5.78 0.15 5.66 0.29 -9% - 5% SpanNear 6.32 0.14 6.40 0.16 -3% - 6% Term 41.60 2.44 38.12 0.33 -14% - -1% TermBGroup1M 14.40 0.48 13.73 0.19 -8% - 0% TermBGroup1M1P 23.68 0.44 22.82 0.44 -7% - 0% TermGroup1M 15.25 0.48 14.51 0.20 -9% - 0% Wildcard 32.76 0.53 31.76 0.62 -6% - 0% {noformat}

55. bq. But the problem is some other exception has already been thrown (because success is false). If out.close then hits a second exception we have to pick which one should be thrown, and I think the original one is better? (Since it's likely the root cause of whatever went wrong). OK, I see, then let's change ForPostingsFormat.fieldsConsumer/Producer as well.
56. OK I committed that! Let me know if I missed any...
57. OK, thanks!
58. Fix a minor bug for maxChain, which will not be detected when blockSize<256. Support numBits=0, however, in wiki data, there are only 0.09% of the blocks encoded as numBits=0, this doesn't actually affect the performance. Also add some testcases.
59. Thanks Billy, I'll commit! One thing I noticed: I think we shouldn't separately read numBytes and the int header? Can't we do a single readVInt(), and that encodes numBytes as well as format (bit width and format, once we tie into oal.util.packed APIs)? Also, we shouldn't encode numInts at all, ie, this should be fixed for the whole segment, and not written per block.
60. I didn't commit lucene/core/src/java/org/apache/lucene/codecs/pfor/ForPostingsFormat.java -- your IDE had changed it to a wildcard import (I prefer we stick with individual imports). Was the numBits==0 case for all 0s not all 1s? We may want to have it mean all 1s instead?
61. bq. Was the numBits==0 case for all 0s not all 1s? We may want to have it mean all 1s instead? OK, I just tested this, and for most cases(93%) when the whole block shares one value v, v==1. This change improves index speed and reduce file size a bit(280s vs 320s and 589M vs 591M). But why? Does lucene store freq() when it is 0 as well, so a whole block with v==1 will be more possible?
62. bq. But why? Does lucene store freq() when it is 0 as well, so a whole block with v==1 will be more possible? A whole block of 1s can easily happen: if all freqs are one (the term always occurred only once in each document), or if the term occurs in every document than the delta between docIDs is always 1. I don't think we should ever hit an all 0s block today (hmm: except for positions, if the given term always occurred at the first position in each doc). We could in theory subtract 1 from all these deltas (except the first one! so maybe we add one to the docID to begin with...) so that these turn into all 0s blocks, but then at decode time we'd have to add 1 back and I'm not sure that'd net/net be a win.
63. bq. We could in theory subtract 1 from all these deltas (except the first one! so maybe we add one to the docID to begin with...) so that these turn into all 0s blocks, but then at decode time we'd have to add 1 back and I'm not sure that'd net/net be a win. Hmm , so current strategy is: 1.for docIDs, store v[i+1]-v[i]-1; 2. for freq and positions, store v[i] directly? Yes there are blocks with all 0s, although very rare to see.
64. No, for docIDs we store docID - lastDocID. So that delta can be 0 for the first doc in a posting list, and then >= 1 thereafter. But an all 0s block is possible if a bunch of terms in a row occurred only in doc 0.
65. This patch cut the extra header and merge numBytes into header. Also, it store only one int when a whole block share the same value. So no matter which strategy we use (d-gap, or d-gap minus 1), it will work well. And here are the changes between different methods, postingsformat=PFor: {noformat} method base all_0s all_1s all_Vs all_Vs+header_cut index time(s): 324 315 279 279 291 index size(MB): 591 591 589 589 577 {noformat} postingsFormat=For: {noformat} method base all_Vs+header_cut index time(s): 250 251 index size(MB): 611 598 {noformat} where raw refers to the version when numBits==0 isn't supported, all_0s refers to last patch, all_Vs+header_cut refers to this patch. As for PFor, now the index size is almost equal to Lucene40(590.7M vs 590.0M).
66. previous patch is a little messy, do some cleanups.
67. Those are interesting results! Curious how much faster indexing is for PFor if you use all_Vs; cutting the header is also a nice reduction on index size. Instead of having P/ForUtil reach up into P/ForPostingsFormat for the default block size, I think we can assume the int[] array length (of the decoded buffer) is the size of the block?
68. bq. Instead of having P/ForUtil reach up into P/ForPostingsFormat for the default block size, I think we can assume the int[] array length (of the decoded buffer) is the size of the block? +1, I'll update the patch
69. Maybe we should cleanup those patches first? The latest LUCENE-3892-for&pfor-with-javadoc.patch should be a baseline for current methods. The patch marked as "iterate numbits" uses previous method to estimate compressed size, ignoring all forced exception, while the other one marked as "slow estimate" will fakely compress the whole block several times, and get the lower bound of our compressed size. Here is a comparison: {noformat} method header_cut iter_numBits slow_estimate index size(M) 577 573 554 index time(s) 275 258 296 {noformat}
70. Thanks Billy, I committed last baseline patch!
71. I opened LUCENE-4225 with a new base PostingsFormat that gives better perf for For than Sep...
72. I think a good thing to explore next is to stop using our own packed ints impl and instead cutover to oal.util.packed? (Since so much effort has gone into making those impls fast). LUCENE-4161 has already

taken a big step towards making them usable ... we should prototype an initial cutover and then iterate?

73. +1 Don't hesitate to tell me if you're missing methods for this issue (I'm thinking at least of bulk int[] read/write, we currently only make it possible with longs).
74. bq. I opened LUCENE-4225 with a new base PostingsFormat that gives better perf for For than Sep... Wow, the result looks great! Quite curious why some queries improve so much, like AndHighHigh. bq. LUCENE-4161 has already taken a big step towards making them usable ... we should prototype an initial cutover and then iterate? Yes, but we should make the PostingsFormat pass test first? Currently it also fails some tests for ForPF.
75. bq. Yes, but we should make the PostingsFormat pass test first? Currently it also fails some tests for ForPF. Uh oh I didn't know tests are failing on the branch: do you have a seed?
76. An initial try with PackedInts in current trunk version. I replaced all the int[] buffer with long[] buffer so we can use the API directly. I don't quite understand the Writer part, so we have to save each long value one by one. However, it is the Reader part we are concerned: {noformat} Task QPS base StdDev base QPS packedStdDev packed Pct diff AndHighHigh 29.60 1.56 23.78 0.51 -25% - -13% AndHighMed 74.68 3.92 53.15 2.31 -35% - -21% Fuzzy1 88.23 1.21 87.13 1.41 -4% - 1% Fuzzy2 30.09 0.45 29.47 0.47 -5% - 1% IntNRQ 41.96 3.88 38.16 2.48 -22% - 6% OrHighHigh 17.56 0.34 15.45 0.15 -14% - -9% OrHighMed 34.71 0.76 30.77 0.53 -14% - -7% PKLookup 111.00 1.90 110.52 1.59 -3% - 2% Phrase 9.03 0.23 7.62 0.41 -22% - -8% Prefix3 123.56 8.42 110.94 5.43 -20% - 1% Respell 102.37 1.11 101.79 1.38 -2% - 1% SloppyPhrase 3.97 0.19 3.52 0.07 -17% - -4% SpanNear 8.24 0.18 7.22 0.25 -17% - -7% Term 45.16 3.15 37.47 2.32 -27% - -5% TermBGroup1M 17.19 1.09 15.86 0.77 -17% - 3% TermBGroup1M1P 23.47 1.66 20.43 1.16 -23% - -1% TermGroup1M 19.20 1.14 17.73 0.84 -16% - 2% Wildcard 42.75 3.27 36.75 1.96 -24% - -1% {noformat} Maybe we should try PACKED_SINGLE_BLOCK for some special value of numBits, instead of using PACKED all the time? Thanks to Adrien, we have a more direct API in LUCENE-4239, I'm trying that now.
77. Patch with the decoder interface, mentioned in LUCENE-4239. I'm afraid that the for loop of readLong() hurts the performance. Here is the comparison against last patch: {noformat} Task QPS base StdDev base QPS comp StdDev comp Pct diff AndHighHigh 21.89 0.64 22.14 0.43 -3% - 6% AndHighMed 52.23 2.34 52.94 1.74 -6% - 9% Fuzzy1 86.61 1.63 87.29 3.14 -4% - 6% Fuzzy2 30.54 0.54 30.95 1.18 -4% - 7% IntNRQ 38.00 1.23 38.14 1.04 -5% - 6% OrHighHigh 16.37 0.21 16.68 0.79 -4% - 8% OrHighMed 39.59 0.69 40.34 2.16 -5% - 9% PKLookup 111.51 1.34 112.78 1.37 -1% - 3% Phrase 4.54 0.12 4.52 0.13 -5% - 5% Prefix3 107.85 2.51 109.13 2.10 -3% - 5% Respell 123.21 2.18 125.15 5.01 -4% - 7% SloppyPhrase 6.51 0.11 6.44 0.29 -7% - 5% SpanNear 5.36 0.16 5.31 0.14 -6% - 4% Term 42.49 1.66 44.10 1.86 -4% - 12% TermBGroup1M 17.86 0.80 17.82 0.51 -7% - 7% TermBGroup1M1P 21.08 0.55 21.10 0.62 -5% - 5% TermGroup1M 19.57 0.82 19.57 0.64 -7% - 7% Wildcard 43.99 1.21 44.80 1.10 -3% - 7% {noformat}
78. {quote} I'm afraid that the for loop of readLong() hurts the performance. Here is the comparison against last patch: {quote} I think so too. I think in each enum, up front you want a pre-allocated byte[] (maximum size possible for the block), and you do ByteBuffer.wrap(x).asLongBuffer. after you read the header, call readBytes() and then just rewind()? So this is just like what you do now in the branch, except with LongBuffer instead of IntBuffer
79. So I changed the patch to readBytes(): base: PackedInts.getReaderNoHeader().get(long[]), file io is handled by PackedInts. comp: PackedInts.getDecoder().decode(LongBuffer,LongBuffer), use byte[] to hold the compressed block, and ByteBuffer.wrap().asLongBuffer as a wrapper. Well, not as expected. {noformat} Task QPS base StdDev base QPS comp StdDev comp Pct diff AndHighHigh 23.78 1.06 23.38 0.42 -7% - 4% AndHighMed 52.06 3.28 50.82 1.21 -10% - 6% Fuzzy1 88.56 0.59 88.98 2.38 -2% - 3% Fuzzy2 28.80 0.36 28.97 0.83 -3% - 4% IntNRQ 41.92 1.67 41.34 0.50 -6% - 3% OrHighHigh 15.85 0.45 15.89 0.39 -4% - 5% OrHighMed 20.38 0.61 20.50 0.62 -5% - 6% PKLookup 110.72 2.19 111.74 2.53 -3% - 5% Phrase 7.51 0.12 7.05 0.18 -9% - -2% Prefix3 106.27 2.65 105.37 1.13 -4% - 2% Respell 112.03 0.81 112.79 2.71 -2% - 3% SloppyPhrase 15.43 0.48 14.92 0.27 -7% - 1% SpanNear 3.52 0.10 3.41 0.06 -7% - 1% Term 39.19 1.34 39.04 0.81 -5% - 5% TermBGroup1M 18.45 0.68 18.33 0.56 -7% - 6% TermBGroup1M1P 22.78 0.90 22.26 0.56 -8% - 4% TermGroup1M 19.50 0.73 19.42 0.63 -7% - 6% Wildcard 29.56 1.13 29.18 0.28 -5% - 3% {noformat}
80. FYI: I committed the TestPostingsFormat here to trunk/4.x to get it going in jenkins. I will merge back to the branch... it can then be modified/improved as usual!
81. Previous experiments showed a net loss with packed ints API, however there're slight difference e.g. all-value-the-same case is not handled equally. I suppose these two patches should make the comparison fair enough. Base: BlockForPF + hardcoded decoder Comp: BlockForPF + PackedInts.Decoder {noformat} Task QPS base StdDev base QPS comp StdDev comp Pct diff AndHighHigh 25.66 0.31 22.61 1.21 -17% - -6% AndHighMed 74.17 1.45 59.48 3.62 -26% - -13% Fuzzy1 95.60 1.51 96.06 2.22 -3% - 4% Fuzzy2

28.67 0.50 28.51 0.75 -4% - 3% IntNRQ 33.31 0.60 30.73 1.51 -13% - -1% OrHighHigh 17.58 0.59 16.22 1.18 -17% - 2% OrHighMed 34.42 0.93 32.14 2.33 -15% - 2% PKLookup 217.08 4.25 213.76 1.37 -4% - 1% Phrase 6.10 0.12 5.34 0.07 -15% - -9% Prefix3 77.27 1.26 70.42 2.87 -13% - -3% Respell 92.91 1.34 92.61 1.83 -3% - 3% SloppyPhrase 5.35 0.16 5.00 0.29 -14% - 1% SpanNear 6.05 0.15 5.47 0.07 -12% - -6% Term 37.62 0.32 33.08 1.70 -17% - -6% TermBGroup1M 17.45 0.64 16.40 0.73 -13% - 1% TermBGroup1M1P 25.20 0.69 23.47 1.24 -14% - 0% TermGroup1M 18.53 0.65 17.40 0.76 -13% - 1% Wildcard 44.39 0.49 40.51 1.69 -13% - -3% {noformat} Hmm, quite strange that we are already getting perf loss with baseline patch: Base: BlockForPF in current branch Comp: BlockForPF + hardcoded decoder(patch file) {noformat} Task QPS base StdDev base QPS comp StdDev comp Pct diff AndHighHigh 26.71 0.98 24.15 0.82 -15% - -2% AndHighMed 73.37 5.01 61.30 1.97 -24% - -7% Fuzzy1 85.73 4.95 84.30 1.79 -9% - 6% Fuzzy2 30.15 2.05 29.52 0.66 -10% - 7% IntNRQ 38.56 1.69 36.91 1.27 -11% - 3% OrHighHigh 16.98 1.48 16.82 0.94 -13% - 14% OrHighMed 34.60 2.79 34.70 2.22 -13% - 16% PKLookup 214.93 3.99 213.86 1.23 -2% - 1% Phrase 11.53 0.23 10.75 0.42 -12% - -1% Prefix3 107.15 3.83 102.12 2.69 -10% - 1% Respell 87.41 5.41 86.08 1.76 -9% - 7% SloppyPhrase 5.90 0.15 5.66 0.21 -9% - 2% SpanNear 4.99 0.12 4.79 0.01 -6% - -1% Term 49.37 2.38 45.53 0.49 -12% - -2% TermBGroup1M 17.23 0.40 16.44 0.53 -9% - 0% TermBGroup1M1P 22.02 0.50 22.42 0.60 -3% - 7% TermGroup1M 13.65 0.29 13.05 0.28 -8% - 0% Wildcard 48.73 2.01 46.35 1.31 -11% - 2% {noformat}

82. I'm confused by these two patches: are they against trunk? How come eg they have mods to build.xml?
83. OK I think I understand the two patches now. First, the build.xml changes are noise I think. Second, the patches both mix in the removal of the current For/PFor postings formats based on sep (I will separately commit this removal: BlockPF is faster). Then, one patch (LUCENE-3892-blockFor&hardcode(base).patch) keeps using the separate packed-ints impl we have, but cuts over to LongBuffer instead of int[] for the decoded values (still uses IntBuffer for the encoded values), while the other patch (LUCENE-3892-blockFor&packedecoder(comp).patch) uses oal.util.packed and LongBuffer for both encoded and decoded values. So it's nice to see that "merely" switching to LongBuffer to pass encoded/decoded values around doesn't seem to hurt much, except for And queries (odd?), but then switching to oal.util.packed does hurt (also odd because our packed ints impl has been heavily optimized lately).
84. My benchmark results are a little different but oal.util.packed is still behind... (it compares the current branch vs. patched with PackedInts): {noformat} TaskQPS pforcodecStdDev pforcodecQPS pforcodec-packedintsStdDev pforcodec-packedints Pct diff Phrase 38.21 3.01 35.73 2.41 -19% - 8% SpanNear 27.99 1.30 26.30 1.23 -14% - 3% SloppyPhrase 43.32 2.98 41.02 2.53 -16% - 7% AndHighMed 230.23 8.48 219.88 9.35 -11% - 3% AndHighHigh 52.53 2.02 50.80 2.62 -11% - 5% IntNRQ 43.24 3.42 41.84 2.79 -16% - 12% Wildcard 113.26 3.17 109.91 3.50 -8% - 3% Prefix3 194.56 9.56 189.39 9.64 -11% - 7% Term 301.86 14.49 295.28 17.51 -12% - 8% OrHighMed 100.60 8.30 99.06 8.00 -16% - 15% OrHighHigh 32.35 2.92 31.90 2.88 -17% - 18% Fuzzy2 36.27 0.67 35.87 0.93 -5% - 3% Fuzzy1 81.14 1.24 80.24 1.68 -4% - 2% TermGroup100K 193.40 3.36 191.27 4.13 -4% - 2% TermBGroup100K1P 152.78 5.06 151.23 3.98 -6% - 5% TermBGroup100K 242.78 7.06 240.71 8.01 -6% - 5% Respell 85.75 1.36 85.17 2.04 -4% - 3% PKLookup 206.02 5.05 205.57 4.63 -4% - 4% {noformat} I am not sure why oal.util.packed is slower. The only differences I see is that they use inheritance instead of a switch block to know how to decode data and that they encode values in the high-order long bits first while the branch currently starts with the low-order int bits. I'll try to dig deeper to understand what happens...
85. I just committed a new BlockPacked postings format, which is a copy of Block postings format but using oal.util.packed for encode/decode. I left Block unchanged, except I moved the util classes it had been using out of oal.codecs.pfor, and removed oal.codecs.pfor. So now we can iterate to speed up packed ints cutover, and do perf tests off the branch.
86. Sorry I meant to say: the BlockPacked PF is from Billy's LUCENE-3892-blockFor&packedecoder(comp).patch.
87. I tested Block vs BlockPacked as checked in. On a Westmere Xeon machine (Java 1.7.0_04): {noformat} Task QPS base StdDev base QPS for StdDev for Pct diff AndHighMed 15.14 0.14 13.78 0.13 -10% - -7% SloppyPhrase 2.55 0.11 2.33 0.09 -15% - -1% OrHighHigh 3.75 0.16 3.44 0.09 -14% - -1% Wildcard 8.44 0.01 7.78 0.28 -11% - -4% SpanNear 1.11 0.04 1.03 0.04 -13% - 0% Prefix3 17.91 0.08 16.63 0.50 -10% - -3% OrHighMed 11.35 0.65 10.63 0.44 -15% - 3% IntNRQ 6.73 0.03 6.32 0.27 -10% - -1% TermBGroup1M 3.87 0.03 3.68 0.04 -6% - -3% AndHighHigh 4.86 0.09 4.63 0.03 -7% - -2% Phrase 1.10 0.06 1.05 0.06 -14% - 6% Term 7.86 0.03 7.52 0.04 -5% - -3% TermBGroup1M1P 4.65 0.12 4.49 0.06 -6% - 0% TermGroup1M 2.97 0.04 2.88 0.02 -4% - -1% Fuzzy1 71.22 1.93 71.02 1.44 -4% - 4% Fuzzy2 49.76 1.33 49.90 1.23 -4% - 5% Respell 76.23 2.67 76.93 2.67 -5% - 8% PKLookup 161.89 3.28 168.28 7.87 -2% - 11% {noformat} And on an desktop Ivy Bridge (Java 1.7.0_04): {noformat} Task QPS base

StdDev base QPS for StdDev for Pct diff AndHighMed 17.32 0.12 15.41 0.03 -11% - -10% SloppyPhrase 2.74 0.21 2.56 0.11 -16% - 5% Phrase 1.32 0.07 1.23 0.06 -15% - 3% Wildcard 9.65 0.11 9.08 0.12 -8% - -3% SpanNear 1.20 0.01 1.13 0.01 -7% - -3% AndHighHigh 5.32 0.03 5.04 0.02 -6% - -4% Prefix3 18.93 0.20 18.04 0.24 -6% - -2% IntNRQ 7.79 0.13 7.48 0.13 -7% - 0% Term 9.48 0.10 9.15 0.43 -8% - 2% TermBGroup1M 4.74 0.05 4.59 0.12 -6% - 0% OrHighMed 13.01 0.24 12.60 0.55 -9% - 2% OrHighHigh 4.08 0.05 3.97 0.17 -8% - 2% TermGroup1M 3.30 0.03 3.22 0.07 -5% - 0% TermBGroup1M1P 5.52 0.11 5.42 0.22 -7% - 4% PKLookup 194.62 4.43 193.44 5.07 -5% - 4% Fuzzy1 79.23 1.31 79.21 0.96 -2% - 2% Respell 78.97 1.04 79.87 1.15 -1% - 3% Fuzzy2 56.17 0.93 56.82 0.64 -1% - 4% {noformat} So packed is still behind ...

88. I just committed an optimization to BlockPF DocsEnum.advance, inlining the scanning step (still have to do D&PEnum and EverythingEnum): {noformat} Task QPS base StdDev base QPS for StdDev for Pct diff IntNRQ 12.46 1.45 11.60 0.04 -16% - 5% Wildcard 54.36 2.75 52.72 0.38 -8% - 2% Prefix3 85.43 4.97 83.08 0.47 -8% - 3% Fuzzy2 63.86 2.13 62.44 1.79 -8% - 4% Respell 62.75 1.52 61.42 2.02 -7% - 3% Fuzzy1 75.68 1.65 74.69 1.44 -5% - 2% LowSpanNear 9.24 0.20 9.13 0.19 -5% - 3% PKLookup 192.89 2.91 190.66 2.43 -3% - 1% HighSpanNear 1.71 0.05 1.69 0.05 -6% - 4% MedSpanNear 4.80 0.11 4.76 0.12 -5% - 4% MedPhrase 12.57 0.27 12.56 0.21 -3% - 3% MedSloppyPhrase 6.57 0.11 6.56 0.11 -3% - 3% LowPhrase 21.55 0.35 21.55 0.28 -2% - 2% LowSloppyPhrase 7.25 0.16 7.28 0.12 -3% - 4% HighPhrase 1.81 0.11 1.82 0.10 -10% - 13% HighSloppyPhrase 1.94 0.10 1.96 0.05 -6% - 9% LowTerm 512.53 5.66 518.31 2.30 0% - 2% MedTerm 196.09 4.68 198.76 0.30 -1% - 3% HighTerm 35.53 0.95 36.11 0.03 -1% - 4% OrHighMed 23.34 0.83 23.85 0.70 -4% - 9% OrHighLow 26.91 0.98 27.53 0.82 -4% - 9% OrHighHigh 11.27 0.41 11.53 0.34 -4% - 9% AndHighHigh 21.24 0.05 23.79 0.13 11% - 12% AndHighLow 553.19 8.47 621.35 4.01 9% - 14% AndHighMed 57.45 0.13 67.78 0.70 16% - 19% {noformat}
89. I backported Mike's changes to the {{BlockPacked}} codec and tried to understand why it was slower than {{Block}}... The use of {{java.nio.*Buffer}} seemed to be the bottleneck ({{ByteBuffer.asLongBuffer}} and {{ByteBuffer.getLong}} especially are _very_ slow) of the decoding step so I switched back to decoding from long[] (instead of LongBuffer) and added direct decoding from byte[] to avoid having to convert the bytes to longs before decoding. Tests passed with -Dtests.postingsformat=BlockPacked. Here are the results of the benchmark (unfortunately, it started before Mike committed r1370179): {noformat} Task QPS 3892 StdDev 3892QPS 3892-packedStdDev 3892-packed Pct diff PKLookup 259.41 9.06 255.77 8.89 -8% - 5% AndHighLow 1656.30 50.44 1653.85 55.05 -6% - 6% AndHighHigh 82.90 1.82 83.47 2.52 -4% - 6% AndHighMed 274.76 11.11 278.51 13.42 -7% - 10% Prefix3 285.41 4.82 289.60 6.31 -2% - 5% HighTerm 230.78 14.33 235.16 20.61 -12% - 18% IntNRQ 55.91 1.03 57.13 2.73 -4% - 9% LowTerm 1720.10 47.06 1759.16 55.47 -3% - 8% Wildcard 290.54 3.82 297.39 5.42 0% - 5% MedTerm 733.01 35.38 750.46 50.37 -8% - 14% HighSpanNear 6.93 0.23 7.12 0.39 -6% - 11% HighPhrase 6.46 0.22 6.65 0.46 -7% - 14% Respell 96.11 2.84 99.00 3.98 -3% - 10% OrHighHigh 38.07 2.53 39.23 3.06 -10% - 19% Fuzzy2 50.29 1.70 51.87 2.25 -4% - 11% MedPhrase 26.20 0.94 27.03 1.07 -4% - 11% OrHighMed 138.83 7.76 143.54 9.79 -8% - 16% Fuzzy1 100.58 2.15 104.21 3.99 -2% - 9% HighSloppyPhrase 5.26 0.11 5.45 0.24 -3% - 10% OrHighLow 78.43 5.55 81.80 6.89 -10% - 21% MedSpanNear 32.75 1.13 34.28 1.73 -3% - 13% LowPhrase 90.27 3.20 95.06 3.58 -2% - 13% LowSpanNear 46.40 1.95 48.89 2.40 -3% - 15% MedSloppyPhrase 36.29 1.00 38.59 1.46 0% - 13% LowSloppyPhrase 37.41 1.11 40.48 1.39 1% - 15% {noformat} Mike, Billy, could you check that {{BLockPacked}} is at least as fast as {{Block}} on your computer too?
90. Thanks Adrien! Your codes are really clean! At first glance, I think we should still support all-value-the-same case? For some applications(like index with payloads), that might be helpful. And, I'm a little confused about your performance test. Did you use BlockPF before r1370179 as a baseline, and compare it with your latest commit? Here, I tested these two PF under latest versions(r1370345). {noformat} Task QPS base StdDev base QPS comp StdDev comp Pct diff AndHighHigh 124.53 9.36 100.46 3.31 -27% - -9% AndHighLow 2141.08 63.93 1922.73 36.32 -14% - -5% AndHighMed 281.48 36.49 218.68 13.10 -35% - -5% Fuzzy1 84.33 2.56 83.94 1.67 -5% - 4% Fuzzy2 30.49 1.13 30.48 0.71 -5% - 6% HighPhrase 9.08 0.28 7.56 0.20 -21% - -11% HighSloppyPhrase 5.46 0.21 4.88 0.23 -17% - -2% HighSpanNear 10.12 0.21 9.21 0.30 -13% - -3% HighTerm 176.52 6.13 146.13 5.43 -22% - -11% IntNRQ 59.56 1.98 51.05 1.33 -19% - -9% LowPhrase 40.02 1.03 32.75 0.37 -21% - -15% LowSloppyPhrase 59.59 2.85 51.49 1.33 -19% - -6% LowSpanNear 73.86 3.17 61.98 1.45 -21% - -10% LowTerm 1755.38 15.56 1622.61 26.87 -9% - -5% MedPhrase 25.99 0.47 21.01 0.17 -21% - -16% MedSloppyPhrase 30.52 0.89 24.77 0.55 -22% - -14% MedSpanNear 22.26 0.43 18.73 0.47 -19% - -12% MedTerm 651.90 18.97 573.34 19.25 -17% - -6% OrHighHigh 26.75 0.33 23.53 0.50 -14% - -9% OrHighLow 151.69 2.13 134.17 3.19 -14% - -8% OrHighMed 102.48 1.48 90.73 2.01 -14% - -8% PKLookup 216.59 5.70 215.99 2.99 -4% - 3% Prefix3 166.00 0.78 145.25 1.29 -13% - -11% Respell 82.01 3.01 82.80 1.66 -4% - 6% Wildcard 151.66 2.22

141.14 1.57 -9% - -4% {noformat} Strange that it isn't working well on my computer. And results are similar when I change MMapDirectory to NIOFSDirectory.

91. Hmm also not great results on my env (base=Block, packed=BlockPacked), based on current branch head: {noformat} Task QPS base StdDev base QPS packedStdDev packed Pct diff AndHighMed 59.23 3.07 34.24 0.69 -46% - -37% AndHighLow 576.35 21.09 349.57 7.44 -42% - -35% AndHighHigh 23.83 0.72 15.53 0.29 -37% - -31% MedPhrase 12.56 0.20 8.87 0.31 -32% - -25% LowPhrase 20.52 0.21 14.89 0.43 -30% - -24% MedSloppyPhrase 7.46 0.20 5.41 0.13 -31% - -23% LowSloppyPhrase 6.73 0.18 4.92 0.12 -30% - -22% LowSpanNear 7.63 0.32 5.65 0.19 -31% - -20% HighSloppyPhrase 1.90 0.08 1.52 0.05 -25% - -14% HighPhrase 1.57 0.04 1.26 0.08 -26% - -12% MedSpanNear 3.84 0.18 3.14 0.14 -25% - -10% LowTerm 433.22 34.89 364.03 15.63 -25% - -4% HighSpanNear 1.40 0.07 1.19 0.06 -23% - -6% IntNRQ 9.50 0.43 8.09 0.92 -27% - 0% HighTerm 29.47 4.89 25.46 2.35 -32% - 13% MedTerm 148.76 21.53 129.17 9.59 -29% - 9% Prefix3 72.81 2.20 63.65 3.88 -20% - -4% Wildcard 44.79 0.92 39.91 2.20 -17% - -4% OrHighMed 16.81 0.48 15.28 0.21 -12% - -5% OrHighLow 21.85 0.67 20.03 0.32 -12% - -3% OrHighHigh 8.49 0.28 7.80 0.14 -12% - -3% Fuzzy1 61.33 1.95 58.91 1.11 -8% - 1% PKLookup 156.87 1.14 154.08 2.13 -3% - 0% Respell 58.72 1.57 59.60 1.28 -3% - 6% Fuzzy2 60.98 2.34 62.03 1.89 -5% - 9% {noformat} I think optimizing the all-values-same case is actually quite important for payloads (but luceneutil doesn't test this today). But, curiously, my BlockPacked index is a bit smaller than my Block index (4643 MB vs 4650 MB). I do wonder about using long[] to hold the uncompressed results (they only need int[]); that's one big difference still. Also: I'd love to see how acceptableOverheadRatio > 0 does ... (and, using PACKED_SINGLE_BLOCK ... we'd have to put a bit in the header to record the format).
92. I tried smaller block sizes than 128. Here's 128 (base) vs 64: {noformat} Task QPS base StdDev base QPS block64StdDev block64 Pct diff AndHighHigh 23.91 0.57 22.28 0.27 -10% - -3% AndHighMed 60.63 1.02 56.96 1.13 -9% - -2% MedSloppyPhrase 7.69 0.01 7.30 0.13 -6% - -3% HighSloppyPhrase 1.93 0.02 1.83 0.04 -8% - -1% LowSloppyPhrase 6.84 0.03 6.57 0.11 -6% - -1% Fuzzy1 65.49 0.85 63.50 1.68 -6% - 0% HighPhrase 1.57 0.04 1.53 0.04 -7% - 3% OrHighLow 22.89 0.98 22.38 0.61 -8% - 4% OrHighMed 17.65 0.70 17.27 0.43 -8% - 4% IntNRQ 9.50 0.48 9.33 0.36 -10% - 7% OrHighHigh 8.98 0.36 8.84 0.19 -7% - 4% HighTerm 29.60 2.64 29.16 1.44 -13% - 13% Fuzzy2 65.54 0.86 64.63 2.13 -5% - 3% Wildcard 45.27 1.27 44.78 0.48 -4% - 2% MedTerm 150.40 12.65 148.99 6.63 -12% - 12% Prefix3 72.55 2.55 72.31 1.02 -5% - 4% LowTerm 421.62 38.27 422.40 9.47 -10% - 12% LowSpanNear 7.55 0.34 7.62 0.22 -6% - 8% HighSpanNear 1.34 0.09 1.35 0.06 -9% - 12% MedPhrase 12.45 0.24 12.66 0.13 -1% - 4% Respell 59.54 1.80 60.95 1.86 -3% - 8% MedSpanNear 3.70 0.24 3.80 0.15 -7% - 14% PKLookup 154.56 2.45 158.96 1.89 0% - 5% LowPhrase 20.21 0.33 20.95 0.15 1% - 6% AndHighLow 577.81 12.46 637.96 29.80 3% - 18% {noformat} And 128 (base) vs 32: {noformat} Task QPS base StdDev base QPS block64StdDev block64 Pct diff AndHighHigh 23.86 0.52 20.68 0.59 -17% - -8% IntNRQ 9.48 0.38 8.84 0.46 -15% - 2% HighSloppyPhrase 1.87 0.04 1.76 0.06 -11% - 0% Prefix3 72.65 2.18 68.24 2.96 -12% - 1% HighTerm 29.91 1.40 28.28 2.94 -19% - 9% Wildcard 44.74 0.83 42.43 1.49 -10% - 0% HighSpanNear 1.37 0.08 1.30 0.07 -15% - 6% MedTerm 152.73 5.28 145.45 14.69 -17% - 8% MedSloppyPhrase 7.46 0.12 7.12 0.25 -9% - 0% HighPhrase 1.57 0.03 1.50 0.01 -7% - -1% OrHighLow 22.94 0.70 22.00 1.10 -11% - 3% AndHighMed 58.72 1.79 56.60 1.95 -9% - 2% LowSloppyPhrase 6.67 0.10 6.44 0.20 -7% - 1% OrHighMed 17.52 0.56 17.00 0.82 -10% - 5% LowSpanNear 7.53 0.35 7.34 0.39 -11% - 7% OrHighHigh 8.84 0.31 8.62 0.43 -10% - 6% MedSpanNear 3.79 0.20 3.71 0.21 -12% - 9% PKLookup 153.34 3.22 150.19 4.91 -7% - 3% Fuzzy1 62.93 1.77 62.28 2.23 -7% - 5% LowTerm 410.23 21.57 410.83 35.19 -13% - 14% MedPhrase 12.55 0.14 12.65 0.08 0% - 2% LowPhrase 20.42 0.17 20.77 0.21 0% - 3% Fuzzy2 61.44 3.12 64.13 1.97 -3% - 13% Respell 56.65 3.29 60.21 1.39 -1% - 15% AndHighLow 588.05 12.37 720.63 19.33 16% - 28% {noformat} It looks like there's some speedup to AndHighLow and LowPhrase ... but slowdowns in other (harder) queries... so I think net/net we should leave block size at 128.
93. Thanks Mike. And detailed comparison result on my computer is here: http://pastebin.com/HLaAuCNp I tried block size range from 1024~32, also used 128 as the base.
94. And result on skipMulitiplier, use current 8 as the baseline: http://pastebin.com/TG4C6u6S Somewhat noisy, but or-queries benifit a little when skipMultiplier=32. And results when we set blockSize fixed to 64: http://pastebin.com/FQBiKGim
95. I tested BulkVInt again, ie to decouple the cutover from Sep to BlockPF vs the vInt/FOR change. Base=Lucene40, comp=BlockPF(BulkVInt): {noformat} Task QPS base StdDev baseQPS bulkVIntStdDev bulkVInt Pct diff AndHighLow 857.35 20.10 614.20 10.73 -31% - -25% Respell 62.99 2.35 60.53 1.34 -9% - 2% AndHighMed 65.64 2.24 63.61 0.93 -7% - 1% Fuzzy2 62.83 1.75 61.72 1.31 -6% - 3% PKLookup 195.97 1.87 194.73 5.00 -4% - 2% IntNRQ 12.50 0.10 12.43 1.49 -13% - 12% Fuzzy1 72.68 1.12 73.84 0.88 -1% - 4% HighPhrase 1.75 0.05 1.78 0.08 -5% - 8% LowSpanNear 9.01

0.12 9.27 0.13 0% - 5% LowPhrase 19.73 0.43 20.64 0.15 1% - 7% MedSpanNear 4.52 0.06 4.74 0.01 3% - 6% MedPhrase 11.74 0.31 12.40 0.09 2% - 9% LowTerm 435.96 13.41 467.22 9.10 1% - 12% Prefix3 75.47 0.51 81.52 4.38 1% - 14% Wildcard 48.66 0.44 52.79 2.79 1% - 15% OrHighHigh 10.11 0.63 11.06 0.32 0% - 20% OrHighMed 20.85 1.31 22.99 0.63 0% - 20% HighSpanNear 1.50 0.02 1.67 0.01 8% - 13% OrHighLow 23.55 1.46 26.51 0.76 2% - 23% LowSloppyPhrase 6.45 0.14 7.37 0.18 9% - 19% MedTerm 163.46 10.30 188.55 5.22 5% - 26% MedSloppyPhrase 5.74 0.12 6.65 0.15 10% - 20% HighSloppyPhrase 1.69 0.04 1.98 0.11 8% - 26% AndHighHigh 19.00 0.53 22.91 0.24 16% - 25% HighTerm 28.28 1.95 34.48 0.99 10% - 34% {noformat} Base=BlockPF(BulkVInt), comp=BlockPF(FOR): {noformat} Task QPS base StdDev base QPS for StdDev for Pct diff IntNRQ 12.10 1.70 11.61 0.02 -16% - 11% HighSloppyPhrase 2.00 0.11 1.95 0.03 -8% - 4% HighPhrase 1.85 0.05 1.81 0.07 -8% - 4% Wildcard 52.32 3.09 52.49 0.24 -5% - 7% LowSloppyPhrase 7.41 0.24 7.43 0.19 -5% - 6% MedSloppyPhrase 6.69 0.18 6.72 0.21 -5% - 6% OrHighMed 22.99 0.55 23.23 0.85 -4% - 7% Respell 61.99 2.01 62.70 1.57 -4% - 7% OrHighLow 26.52 0.69 26.83 1.00 -5% - 7% Fuzzy1 74.72 1.34 75.59 1.43 -2% - 4% PKLookup 189.68 7.14 192.09 3.82 -4% - 7% OrHighHigh 11.05 0.27 11.21 0.42 -4% - 7% Fuzzy2 62.78 1.86 63.70 1.87 -4% - 7% HighSpanNear 1.65 0.03 1.69 0.02 0% - 5% Prefix3 80.25 5.44 82.57 1.03 -4% - 11% AndHighHigh 22.79 0.11 23.53 0.13 2% - 4% LowSpanNear 9.16 0.26 9.48 0.21 -1% - 8% MedSpanNear 4.67 0.09 4.84 0.07 0% - 7% MedPhrase 12.59 0.26 13.07 0.24 0% - 7% LowPhrase 20.86 0.33 22.06 0.30 2% - 8% AndHighLow 618.27 13.15 655.52 3.30 3% - 8% HighTerm 33.95 1.11 36.02 0.08 2% - 9% MedTerm 186.09 5.51 198.46 0.09 3% - 9% AndHighMed 63.71 1.15 69.15 0.45 5% - 11% LowTerm 469.17 7.25 514.55 2.83 7% - 12% {noformat} So ... most of the gains come from BlockPF cutover. This is sort of ... surprising/disappointing, ie, our bottlenecks are the abstraction layers, not the actual decode cost. Still it's good to make progress on removing the abstractions. Also, it looks like the only query that is slower than Lucene40 is AndHighLow ... however, it's also an extremely fast query to begin with so I think it's a fine tradeoff that it gets slower while the hard/slower queries get faster.

96. {quote} So ... most of the gains come from BlockPF cutover. This is sort of ... surprising/disappointing, ie, our bottlenecks are the abstraction layers, not the actual decode cost. Still it's good to make progress on removing the abstractions. {quote} I don't think its that disappointing. This isnt a very interesting benchmark for a compression algorithm like FOR: instead imagine the very common case of apps today indexing small fields like product names, restaurant names, or something like that. Freqs are nearly always 1, and positions are tiny, but often people still want the ability to use things like phrase queries. And imagine cases where people are indexing data from a database and there are only a few unique values (e.g. product type = tshirt, pants, shoes) in a field. I think the wikipedia benchmark doesn't do a very good job of illustrating performance on use-cases like this, which I think are common and also where I'm fairly positive FOR will be a win. Its nice that its not slower or too much bigger in the "worst case" of large docs where the numbers aren't so tiny? {quote} Also, it looks like the only query that is slower than Lucene40 is AndHighLow ... however, it's also an extremely fast query to begin with so I think it's a fine tradeoff that it gets slower while the hard/slower queries get faster. {quote} +1, lets not even think twice about that one.
97. I did some changes to the {{BlockPacked}} codec: - encoding and decoding using int[] instead of long[] - selection of the format based on a configurable overhead ratio. The results are encouraging (using acceptableOverheadRatio = PackedInts.DEFAULT = 20%): {noformat} Task QPS 3892 StdDev 3892QPS 3892-packedStdDev 3892-packed Pct diff PKLookup 256.93 8.89 256.85 7.47 -6% - 6% OrHighLow 145.14 9.86 145.14 9.35 -12% - 14% Respell 110.26 1.84 110.27 2.01 -3% - 3% AndHighHigh 112.97 0.81 113.19 2.17 -2% - 2% Fuzzy1 102.15 1.47 102.86 3.13 -3% - 5% OrHighHigh 94.56 6.56 95.43 6.35 -11% - 15% Fuzzy2 42.49 0.77 42.89 1.43 -4% - 6% OrHighMed 175.30 11.34 177.42 10.83 -10% - 14% AndHighLow 1925.02 23.92 1952.57 48.68 -2% - 5% HighPhrase 8.96 0.41 9.11 0.46 -7% - 11% Wildcard 189.79 2.13 193.12 1.57 0% - 3% HighSpanNear 6.47 0.15 6.59 0.25 -4% - 8% Prefix3 256.67 2.58 262.40 2.84 0% - 4% LowTerm 1746.52 52.80 1789.54 54.30 -3% - 8% HighTerm 238.70 13.46 245.63 16.60 -9% - 16% MedTerm 923.64 38.19 951.18 46.85 -5% - 12% AndHighMed 364.46 3.65 377.09 10.03 0% - 7% IntNRQ 56.58 1.02 58.84 0.80 0% - 7% HighSloppyPhrase 11.73 0.30 12.40 0.62 -2% - 13% LowSpanNear 29.64 0.96 32.44 0.98 2% - 16% MedSpanNear 22.96 0.72 25.16 0.85 2% - 16% MedPhrase 40.99 1.25 45.09 1.24 3% - 16% LowSloppyPhrase 37.88 0.99 41.98 1.49 4% - 17% LowPhrase 64.40 2.04 71.84 1.41 5% - 17% MedSloppyPhrase 42.29 1.16 47.32 1.54 5% - 18% {noformat} I hope this will be confirmed on your computers this time .:-)
98. I also see (smaller) gains with BlockPacked vs Block (this is 10M doc index): {noformat} Task QPS base StdDev base QPS packedStdDev packed Pct diff AndHighMed 69.19 0.53 66.43 0.63 -5% - -2% Fuzzy2 63.71 1.24 62.25 1.58 -6% - 2% Respell 62.69 1.41 61.53 1.47 -6% - 2% IntNRQ 11.86 0.43 11.73 0.03

-4% - 2% Fuzzy1 75.48 1.21 75.05 1.52 -4% - 3% Wildcard 53.23 0.63 52.96 0.25 -2% - 1% MedSpanNear 4.88 0.16 4.88 0.11 -5% - 5% PKLookup 191.48 2.84 191.62 3.98 -3% - 3% HighTerm 35.71 0.63 35.91 0.06 -1% - 2% Prefix3 83.14 1.34 83.83 0.49 -1% - 3% LowTerm 513.35 0.77 517.92 1.50 0% - 1% HighSpanNear 1.70 0.06 1.71 0.03 -4% - 6% AndHighHigh 23.45 0.09 23.69 0.10 0% - 1% OrHighLow 27.27 1.06 27.59 0.15 -3% - 5% OrHighMed 23.61 0.92 23.89 0.17 -3% - 6% OrHighHigh 11.42 0.44 11.59 0.12 -3% - 6% MedSloppyPhrase 6.84 0.17 6.95 0.23 -4% - 7% LowPhrase 22.02 0.39 22.43 0.15 0% - 4% MedTerm 196.76 3.01 200.62 0.33 0% - 3% LowSpanNear 9.60 0.24 9.82 0.31 -3% - 8% MedPhrase 13.08 0.30 13.41 0.12 0% - 5% LowSloppyPhrase 7.55 0.21 7.77 0.27 -3% - 9% AndHighLow 649.84 18.26 669.08 6.63 0% - 6% HighSloppyPhrase 1.98 0.08 2.04 0.09 -4% - 12% HighPhrase 1.76 0.11 1.96 0.10 0% - 24% {noformat} The index is 4669 MB with Block and 4790 with BlockPacked = ~2.6% larger ... seems worth it! Apps can always tune the 20% too.

99. I created a non-specialized (ie single method to handle all numBits cases) packed int decoder that decodes directly from byte[]. Baseline is current BlockPF (FOR w/ specialized decoder), comp is w/ the patch (using non-specialized decoder): {noformat} Task QPS base StdDev base QPS for StdDev for Pct diff AndHighMed 69.04 0.77 36.41 1.91 -50% - -43% AndHighLow 649.70 17.03 346.71 18.22 -50% - -42% LowSpanNear 9.88 0.25 5.53 0.06 -45% - -42% MedPhrase 13.25 0.26 7.74 0.07 -43% - -39% LowSloppyPhrase 7.59 0.15 4.54 0.13 -43% - -37% LowPhrase 22.29 0.31 13.77 0.08 -39% - -36% AndHighHigh 23.55 0.12 15.22 0.63 -38% - -32% MedSloppyPhrase 6.88 0.12 4.60 0.16 -36% - -29% HighSloppyPhrase 1.98 0.07 1.38 0.05 -35% - -25% HighTerm 36.11 0.01 25.31 0.87 -32% - -27% MedSpanNear 5.02 0.16 3.56 0.03 -31% - -26% MedTerm 198.76 0.34 142.92 4.34 -30% - -25% HighPhrase 1.83 0.08 1.32 0.02 -31% - -23% OrHighLow 27.32 1.10 20.55 0.54 -29% - -19% OrHighMed 23.65 0.93 17.83 0.44 -29% - -19% OrHighHigh 11.42 0.46 8.72 0.20 -28% - -18% HighSpanNear 1.74 0.06 1.38 0.01 -24% - -17% IntNRQ 11.61 0.01 9.26 0.02 -20% - -20% LowTerm 513.60 2.26 411.60 7.65 -21% - -18% Prefix3 82.36 1.05 67.48 1.29 -20% - -15% Wildcard 52.63 0.44 43.45 0.81 -19% - -15% Fuzzy1 74.74 1.02 70.03 0.80 -8% - -3% PKLookup 192.60 3.94 191.87 2.07 -3% - 2% Fuzzy2 62.50 1.29 62.74 1.10 -3% - 4% Respell 61.69 1.04 62.79 0.84 -1% - 4% {noformat} So... is it's clear all our the specializing does help!
100. Thanks Mike for your tests. Do you think {{BlockPacked}} is now fast enough to replace {{Block}} with {{BlockPacked}}? I am asking because it is a little painful to always have to backport changes from one format to the other.
101. Yes I think we should do a hard cutover now? Ie, merge any final changes (sorry for all the commits! we are nearly ready to land I think...) over to BlockPacked, then remove Block and rename BlockPacked to Block?
102. Sounds good. I think the only commits that have not been merged yet are 1371010 and 1371011.
103. OK I'll merge & replace Block w/ BlockPacked... likely sometime tomorrow. Thanks Adrien!
104. The comment you added in 1371011 on the value of {{BLOCK_SIZE}} caught my attention: I think that BLOCK_SIZE should be at least 64 with PackedInts encoding/decoding since these conversions are long-aligned (I backported your two commits and added a comment about this). For example, the {{PACKED}} 7-bits encoder cannot encode less than 64 values in one iteration. In case someone would really want to use smaller block sizes (eg. 32), I think it should still perform pretty well if {{acceptableOverheadRatio >= ~25%}} (in that case, all bits-per-value in the [1-24] range either use a {{PACKED_SINGLE_BLOCK}} encoder or an 8-bits, 16-bits or 24-bits {{PACKED}} encoder). Do we plan to make the block size configurable?
105. Thanks Adrien. So now we just have to replace Block with BlockPacked right? OK let's just fix the comment to be multiple of 64. I don't think we need to make BLOCK_SIZE configurable.
106. bq. So now we just have to replace Block with BlockPacked right? Yes, I think so. bq. I don't think we need to make BLOCK_SIZE configurable. In that case, should we also hard-code the value of {{acceptableOverheadRatio}}?
107. Actually let's hold off a bit on replacing Block w/ BlockPacked: Billy was going to do some more tests with PFOR... bq. In that case, should we also hard-code the value of acceptableOverheadRatio? Hmm that one seems more compelling to let apps change?
108. Shouldn't MIN_ENCODED_SIZE be MAX_ENCODED_SIZE? Ie the max number of bytes encoding will ever require. And I think the same for MIN -> MAX_DATA_SIZE? Or maybe MIN_REQUIRED_XXX? I think readVIntBlock shouldn't be in ForUtil? Ie it's very postings-format-specific and it's not using packed ints at all. Also the "equivalent" readVIntBlock code for the positions case (in the readPositions methods) is still in the BlockPackedPostingsReader. I think it's great to have writeBlock/readBlock/skipBlock in ForUtil. Do we really need to write/write the 32 format.getId(), numBits into the postings file header? I guess it's either that or ... store the float acceptableOverheadRatio (eg using Float.floatToIntBits I guess) and have some back-compat enforced in the logic in

PackedInts.fastestFormatAndBits... hmm. Hmm ... MIN_DATA_SIZE is 147 (PACKED_SINGLE_BLOCK, bpv=3), but BLOCK_SIZE is 128 ... so I guess this means if we ever pick that format (because acceptableOverheadRatio allowed us to), we're encoding/decoding those extra 19 unused ints right? (I was just trying to understand why we alloc all the int[] to MIN_DATA_SIZE not BLOCK_SIZE...). ForUtil.getMinRequiredBufferSize seems like dead code?

109. Thank you Adrien! The BlockPacked PF also worked well on my computer :) {noformat} Task QPS base StdDev base QPS packedStdDev packed Pct diff AndHighHigh 122.57 3.01 123.90 2.49 -3% - 5% AndHighLow 2260.53 21.18 2273.77 55.09 -2% - 3% AndHighMed 328.01 8.18 329.31 11.36 -5% - 6% Fuzzy1 86.37 0.94 86.24 2.12 -3% - 3% Fuzzy2 31.40 0.46 31.22 0.64 -4% - 2% HighPhrase 9.09 0.51 9.15 0.40 -8% - 11% HighSloppyPhrase 5.30 0.25 5.34 0.08 -5% - 7% HighSpanNear 10.11 0.44 10.42 0.34 -4% - 11% HighTerm 179.43 7.26 178.96 5.70 -7% - 7% IntNRQ 61.87 3.79 60.59 4.31 -14% - 11% LowPhrase 41.23 1.54 42.97 1.32 -2% - 11% LowSloppyPhrase 62.83 2.11 68.23 0.99 3% - 14% LowSpanNear 81.28 2.74 85.74 2.67 -1% - 12% LowTerm 1763.70 29.21 1778.41 23.07 -2% - 3% MedPhrase 27.06 1.16 27.54 0.88 -5% - 9% MedSloppyPhrase 31.82 1.16 33.70 0.14 1% - 10% MedSpanNear 23.09 0.93 23.84 0.79 -4% - 11% MedTerm 659.09 22.65 671.54 19.79 -4% - 8% OrHighHigh 27.36 0.52 27.41 1.25 -6% - 6% OrHighLow 154.99 2.07 156.20 7.08 -5% - 6% OrHighMed 105.13 1.52 105.30 4.65 -5% - 6% PKLookup 210.64 6.95 217.57 2.08 0% - 7% Prefix3 170.22 6.22 166.80 4.18 -7% - 4% Respell 83.96 1.47 83.75 1.25 -3% - 3% Wildcard 155.08 4.31 155.31 3.12 -4% - 5% {noformat}
110. I think, for a fair test, we should also test w/ acceptableOverheadRatio=0 ... I'll run that.
111. bq. Shouldn't MIN_ENCODED_SIZE be MAX_ENCODED_SIZE? I prefixed with "MIN" because it is the minimum size the encoded buffer size must have to be able to handle all cases. But I think you are right, "MAX" or "REQUIRED" would be clearer. bq. I think readVIntBlock shouldn't be in ForUtil? I'll move it back to BlockPackedPostingsReader. {quote} Do we really need to write/write the 32 format.getId(), numBits into the postings file header? I guess it's either that or ... store the float acceptableOverheadRatio (eg using Float.floatToIntBits I guess) and have some back-compat enforced in the logic in PackedInts.fastestFormatAndBits... hmm.{quote} I hesitated between these two approaches but I think writing all cases to the header is less error-prone? Moreover it would allow us to change the logic of {{fastestFormatAndBits}} without having to bump the version number. {quote} Hmm ... MIN_DATA_SIZE is 147 (PACKED_SINGLE_BLOCK, bpv=3), but BLOCK_SIZE is 128 ... so I guess this means if we ever pick that format (because acceptableOverheadRatio allowed us to), we're encoding/decoding those extra 19 unused ints right? (I was just trying to understand why we alloc all the int[] to MIN_DATA_SIZE not BLOCK_SIZE...).{quote} Exactly. The other problem is that we are also storing these unnecessary 19 values (but it is not easy to fix since PACKED_SINGLE_BLOCK writes values in the low-order long bits first (little endian)). Maybe we should make PACKED_SINGLE_BLOCK write values in the high-order bits first and split byte encoders and decoders from the long ones (so that they have a lower {{valueCount()}}). bq. ForUtil.getMinRequiredBufferSize seems like dead code? I'll remove it.
112. I revived the PFor codes, and test it agains BlockFor and BlockPacked: BlockFor as base: {noformat} Task QPS base StdDev base QPS pfor StdDev pfor Pct diff AndHighHigh 121.54 1.37 116.69 2.03 -6% - -1% AndHighLow 2286.36 14.19 2212.92 11.48 -4% - -2% AndHighMed 322.97 7.37 294.19 4.76 -12% - -5% Fuzzy1 85.56 1.46 87.97 3.27 -2% - 8% Fuzzy2 30.94 0.56 32.16 1.34 -2% - 10% HighPhrase 9.39 0.38 9.02 0.45 -12% - 5% HighSloppyPhrase 5.38 0.08 5.24 0.12 -6% - 1% HighSpanNear 10.38 0.39 9.92 0.08 -8% - 0% HighTerm 180.30 6.87 172.83 6.26 -11% - 3% IntNRQ 62.01 3.73 60.89 3.54 -12% - 10% LowPhrase 42.44 0.67 38.73 0.89 -12% - -5% LowSloppyPhrase 62.82 0.79 56.79 0.43 -11% - -7% LowSpanNear 81.79 2.00 74.10 1.13 -12% - -5% LowTerm 1763.95 39.62 1721.30 34.22 -6% - 1% MedPhrase 27.87 0.59 25.82 0.74 -11% - -2% MedSloppyPhrase 32.15 0.41 29.91 0.31 -9% - -4% MedSpanNear 23.48 0.71 22.00 0.05 -9% - -3% MedTerm 662.11 24.22 638.81 19.31 -9% - 3% OrHighHigh 26.82 0.47 27.14 1.93 -7% - 10% OrHighLow 152.40 3.54 156.58 11.11 -6% - 12% OrHighMed 103.20 2.26 105.84 7.55 -6% - 12% PKLookup 216.38 4.32 219.32 2.59 -1% - 4% Prefix3 169.89 4.97 163.82 3.34 -8% - 1% Respell 83.23 1.44 86.20 3.00 -1% - 9% Wildcard 155.81 2.79 152.30 2.54 -5% - 1% {noformat} BlockPacked as base: {noformat} Task QPS base StdDev base QPS pfor StdDev pfor Pct diff AndHighHigh 122.94 3.43 116.24 1.90 -9% - -1% AndHighLow 2294.32 58.32 2199.14 31.97 -7% - 0% AndHighMed 325.55 12.44 290.20 3.80 -15% - -6% Fuzzy1 88.33 1.84 87.86 2.54 -5% - 4% Fuzzy2 31.92 0.80 32.00 0.92 -5% - 5% HighPhrase 9.73 0.47 9.04 0.29 -14% - 0% HighSloppyPhrase 5.49 0.19 5.16 0.03 -9% - -1% HighSpanNear 10.93 0.23 9.90 0.09 -12% - -6% HighTerm 178.31 6.37 171.06 6.14 -10% - 3% IntNRQ 60.87 4.71 62.38 5.49 -13% - 20% LowPhrase 44.97 1.18 38.36 1.01 -19% - -10% LowSloppyPhrase 69.61 1.19 55.90 1.39 -23% - -16% LowSpanNear 88.50 0.66 72.80 2.23 -20% - -14% LowTerm 1769.84 32.66 1717.02 39.75 -6% - 1% MedPhrase 28.88

0.84 25.57 0.68 -16% - -6% MedSloppyPhrase 34.47 0.50 29.29 0.54 -17% - -12% MedSpanNear 24.88 0.32 21.69 0.38 -15% - -10% MedTerm 667.95 21.61 633.73 22.17 -11% - 1% OrHighHigh 27.96 1.29 26.82 0.81 -11% - 3% OrHighLow 158.62 5.82 155.08 5.05 -8% - 4% OrHighMed 107.16 4.19 104.81 3.17 -8% - 4% PKLookup 217.22 1.86 216.83 1.87 -1% - 1% Prefix3 167.32 6.72 166.12 6.53 -8% - 7% Respell 85.25 2.27 85.85 2.16 -4% - 6% Wildcard 156.24 5.69 154.63 3.02 -6% - 4% {noformat} Current PFor impl only saves 1.8% against For, but get quite some perf loss. Let's use the Packed version!

113. I compared Block w/ BlockPacked, but set acceptableOverheadRatio to 0 for a fairer test: {noformat} Task QPS base StdDev base QPS pack StdDev pack Pct diff HighSloppyPhrase 1.94 0.01 1.91 0.05 -4% - 2% LowPhrase 21.05 0.07 20.84 0.37 -3% - 1% MedPhrase 13.05 0.04 12.93 0.23 -3% - 1% Wildcard 43.87 2.76 43.49 2.10 -11% - 10% IntNRQ 8.88 1.39 8.83 0.78 -21% - 28% Fuzzy1 63.07 1.96 62.78 1.46 -5% - 5% LowSloppyPhrase 6.92 0.01 6.91 0.13 -2% - 1% Prefix3 71.38 5.20 71.35 3.17 -10% - 12% PKLookup 157.00 1.78 158.01 2.01 -1% - 3% AndHighLow 668.76 4.82 674.80 7.48 0% - 2% HighPhrase 1.56 0.03 1.58 0.03 -3% - 5% MedSloppyPhrase 7.71 0.03 7.80 0.11 0% - 2% AndHighMed 74.05 0.49 75.35 0.36 0% - 2% AndHighHigh 25.92 0.30 26.78 0.19 1% - 5% Respell 57.07 2.70 59.20 1.80 -3% - 12% Fuzzy2 60.81 2.92 63.32 1.68 -3% - 12% OrHighHigh 8.99 0.17 9.39 0.11 1% - 7% OrHighMed 17.65 0.37 18.52 0.13 2% - 7% MedSpanNear 3.90 0.17 4.11 0.09 -1% - 12% OrHighLow 22.99 0.51 24.22 0.15 2% - 8% HighSpanNear 1.40 0.06 1.48 0.03 0% - 12% LowSpanNear 7.84 0.31 8.32 0.17 0% - 12% LowTerm 406.02 28.53 444.21 37.75 -6% - 27% MedTerm 149.83 8.11 167.60 15.06 -3% - 28% HighTerm 29.57 1.67 33.42 3.20 -3% - 31% {noformat} Curiously it seems even faster than w/ acceptableOverheadRatio=0.2! But it makes it clear we should do a hard cutover.
114. bq. I revived the PFor codes, and test it agains BlockFor and BlockPacked Thanks Billy, I'll run a test too ...
115. bq. Curiously it seems even faster than w/ acceptableOverheadRatio=0.2! But it makes it clear we should do a hard cutover. I had been doing some tests with the bulk version of PackedInts.get (which uses the same methods that we use for BlockPacked) while working on LUCENE-4098 and it seemed that the bottleneck was more memory bandwidth than CPU (for large arrays at least). If you look at the last graph of http://people.apache.org/~jpountz/packed_ints3.html, the throughput seems to depend more on the memory efficiency of the picked impl than on the way it stores data. Maybe we are experiencing a similar phenomenon here... Unless I am missing something, the only difference between BlockPacked and Block is that BlockPacked decodes directly from byte[] whereas Block uses ByteBuffer.asLongBuffer to translate from bytes to ints and then decodes from the ints... Interesting to know it has so much overhead...
116. OK indeed PFOR is slower for me too: {noformat} Task QPS base StdDev base QPS pfor StdDev pfor Pct diff HighPhrase 1.56 0.03 1.25 0.12 -28% - -10% MedPhrase 13.05 0.10 10.50 0.58 -24% - -14% LowPhrase 21.08 0.08 17.35 0.85 -22% - -13% AndHighMed 73.78 0.66 62.50 1.68 -18% - -12% AndHighLow 674.60 2.54 573.00 12.06 -17% - -12% LowSpanNear 8.04 0.17 6.97 0.23 -17% - -8% MedSpanNear 3.97 0.10 3.58 0.15 -15% - -3% MedSloppyPhrase 7.58 0.11 6.93 0.14 -11% - -5% AndHighHigh 25.71 0.47 23.58 0.61 -12% - -4% HighSpanNear 1.42 0.04 1.31 0.05 -12% - -1% MedTerm 155.44 18.75 144.46 12.33 -24% - 14% HighTerm 30.27 4.31 28.25 2.88 -26% - 19% LowSloppyPhrase 6.73 0.13 6.28 0.12 -10% - -3% OrHighHigh 9.06 0.24 8.53 0.33 -11% - 0% OrHighLow 23.09 0.67 21.88 0.91 -11% - 1% OrHighMed 17.71 0.51 16.79 0.67 -11% - 1% HighSloppyPhrase 1.88 0.05 1.80 0.04 -9% - 0% IntNRQ 9.42 0.50 9.05 0.89 -17% - 11% Prefix3 72.67 2.42 70.42 3.61 -11% - 5% Fuzzy1 63.71 1.07 62.34 1.55 -6% - 1% Wildcard 45.25 0.99 44.28 1.55 -7% - 3% PKLookup 159.04 2.13 157.17 1.90 -3% - 1% Fuzzy2 62.51 2.28 63.40 1.65 -4% - 8% LowTerm 400.06 57.60 407.73 52.40 -22% - 34% Respell 56.72 3.19 59.83 2.10 -3% - 15% {noformat} I think we should replace Block with BlockPacked now?
117. bq. I had been doing some tests with the bulk version of PackedInts.get (which uses the same methods that we use for BlockPacked) while working on LUCENE-4098 and it seemed that the bottleneck was more memory bandwidth than CPU (for large arrays at least). Ahh, interesting... So I think we should test different acceptableOverheadRatios to find the best ... it could be it's 0!
118. {quote} bq. Do we really need to write/write the 32 format.getId(), numBits into the postings file header? I guess it's either that or ... store the float acceptableOverheadRatio (eg using Float.floatToIntBits I guess) and have some back-compat enforced in the logic in PackedInts.fastestFormatAndBits... hmm. I hesitated between these two approaches but I think writing all cases to the header is less error-prone? Moreover it would allow us to change the logic of fastestFormatAndBits without having to bump the version number. {quote} Maybe for starters we should just hardwire acceptableOverheadRatio at 0 ... then we simplify this back-compat until/unless we really need to make this configurable.
119. bq. The other problem is that we are also storing these unnecessary 19 values (but it is not easy to fix since PACKED_SINGLE_BLOCK writes values in the low-order long bits first (little endian)). Maybe

we should make PACKED_SINGLE_BLOCK write values in the high-order bits first and split byte encoders and decoders from the long ones (so that they have a lower valueCount()). OK, we can explore that later (another reason to simply always use Format.PACKED for now...).

120. {quote} OK indeed PFOR is slower for me too: {quote} I think for starters since you guys have gotten FOR pretty nice we should just focus on that one? We could later see if PFOR could get additional wins as a second step: getting FOR working nice and fast is awesome on its own!
121. bq. I think for starters since you guys have gotten FOR pretty nice we should just focus on that one? Yeah I think we should do that. I think the branch is nearly ready to land! I just replaced Block with BlockPacked ...
122. I ran the comparison between acceptableOverheadRatio=PackedInts.COMPACT (0%) and PackedInts.DEFAULT (20%) and it seems to be much faster with PackedInts.COMPACT: {noformat} base=COMPACT, challenger=DEFAULT Task QPS base StdDev base QPS def StdDev def Pct diff IntNRQ 81.83 5.43 74.14 2.94 -18% - 0% HighTerm 146.55 10.34 133.57 9.02 -20% - 4% LowPhrase 93.91 1.63 86.90 1.67 -10% - -4% MedTerm 824.58 43.48 766.35 38.78 -16% - 3% LowSloppyPhrase 83.29 1.99 77.65 1.18 -10% - -3% OrHighMed 94.15 5.28 88.34 4.54 -15% - 4% OrHighHigh 100.63 5.42 94.57 4.20 -14% - 3% OrHighLow 128.62 7.21 120.92 6.07 -15% - 4% HighPhrase 13.05 0.45 12.29 0.39 -11% - 0% Prefix3 217.06 6.82 205.05 4.62 -10% - 0% MedPhrase 27.50 0.97 26.33 0.79 -10% - 2% Wildcard 183.20 4.87 175.58 3.89 -8% - 0% LowTerm 1763.31 43.24 1693.31 39.29 -8% - 0% HighSloppyPhrase 10.05 0.48 9.67 0.40 -11% - 5% AndHighHigh 111.59 1.15 107.45 1.66 -6% - -1% LowSpanNear 56.16 1.32 54.25 1.01 -7% - 0% AndHighMed 423.44 7.40 409.32 5.10 -6% - 0% MedSpanNear 33.14 0.91 32.32 0.74 -7% - 2% AndHighLow 2177.50 30.79 2134.05 28.64 -4% - 0% Fuzzy1 95.34 2.41 93.66 2.32 -6% - 3% HighSpanNear 5.28 0.17 5.21 0.11 -6% - 3% MedSloppyPhrase 18.41 0.72 18.19 0.70 -8% - 6% Fuzzy2 37.73 1.31 37.31 1.14 -7% - 5% Respell 109.71 3.09 108.64 2.76 -6% - 4% PKLookup 257.32 6.64 260.00 7.15 -4% - 6% {noformat}
123. bq. (From mailing-list) So I think if its this ambiguous for wikipedia we should shoot for the most COMPACT form as a safe default. +1 too. I just committed the change.
124. Uwe just started builds for this branch (thanks!): http://jenkins.sd-datasolutions.de/job/pforcodec-3892-branch
125. Patch from Billy improving javadocs ...
126. Patch, applyable to trunk (use patch -p1 < ...). The branch builds look stable ... I think this is ready to land on trunk! I think we should leave Lucene40 as default PF for now, until we BlockPF bakes on trunk for a while, but as some point (maybe for 4.1?) I think we should cutover to BlockPF as the default.
127. From r1373332: {quote} - private static final int PACKED_INTS_VERSION = 0; // nocommit: encode in the stream? + private static final int PACKED_INTS_VERSION_START = 0; + private static final int PACKED_INTS_VERSION_CURRENT = PACKED_INTS_VERSION_START; {quote} Mike, is there any reason why you didn't use {{PackedInts.VERSION_START}} and {{PackedInts.VERSION_CURRENT}} instead?
128. bq. Mike, is there any reason why you didn't use PackedInts.VERSION_START and PackedInts.VERSION_CURRENT instead? Woops, no, I forgot we had version info in PackedInts! I'll switch it over.
129. OK I committed that, and also added version checking in getEncoder/Decoder, and I now loop over all versions when computing MAX_DATA_SIZE -- can you double check Adrien? Thanks!
130. You toasted me, I was just doing exactly the same change! :-) The diff looks good to me.
131. Woops sorry!
132. {quote} The branch builds look stable ... I think this is ready to land on trunk! I think we should leave Lucene40 as default PF for now, until we BlockPF bakes on trunk for a while, but as some point (maybe for 4.1?) I think we should cutover to BlockPF as the default. {quote} +1
133. bq. Woops sorry! NP, good to know we had planned the same changes. bq. The branch builds look stable ... I think this is ready to land on trunk! +1 too
134. I just merged to trunk & 4.x. Thanks Billy, this is a great addition to Lucene!
135. Thank you Mike! And thanks to all of you! I learnt really much this summer!
136. Thanks Billy for all the hard work and endless benchmarking, so nice to have a block codec that is simple and clean and reuses our packed ints optimizations.
137. bq. Thanks Billy for all the hard work and endless benchmarking, so nice to have a block codec that is simple and clean and reuses our packed ints optimizations. +1
138. I tried changing the generated bulk decode methods to re-use variables so we have much fewer local variables required (patch attached), but on quick testing it didn't seem to make much difference in performance.

139. We should keep the size of methods small, as bigger methods work against the code cache of hotspot and if Lucene is not used alone, may get de-optimized.
140. Closed after release.