Item 79
git_comments:

git_commits:

1. **summary:** LUCENE-1421: Update Maven configuration to support the new grouping contrib
   message: LUCENE-1421: Update Maven configuration to support the new grouping contrib git-svn-id: https://svn.apache.org/repos/asf/lucene/dev/branches/branch_3x@1103170 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Ability to group search results by field
   description: It would be awesome to group search results by specified field. Some functionality was provided for Apache Solr but I think it should be done in Core Lucene. There could be some useful information like total hits about collapsed data like total count and so on. Thanks, Artyom

jira_issues_comments:

1. **body:** This is an initial patch that allows result grouping with Lucene via a Collector and an attempt to integrate result grouping into Lucene / Solr. The collector can be used just like any other collector and returns TopDocs. The TopDocs contains GroupDoc instances, which is a subclass of ScoreDoc. I think this way it is easier to integrate grouping into existing code that uses Lucene (like Solr). I think that grouping code should be part of Lucene instead of Solr. I put the result grouping into a new contrib that I named grouping. Putting it in a contib seemed the right place for me. The patch doesn't contain any Solr code and I think a new issue in Solr should be opened for that. This patch is 'inspired by' by SOLR-236, but only contains its core functionality. Nonadjacent grouping based on field value with group counts. Also in the code i don't use the verb collapsing but grouping. This patch is also faster then the Solr variants. This because the grouping occurs whilst the documents are collected and thus saves multiple searches. Also the grouping algorithm itself is improved. Although this is work in progress any thought about this would be appriciated. BTW the patch is based on the Lucene trunk and is relative to the trunk directory
   label: code-design
2. bq. I think that grouping code should be part of Lucene instead of Solr. +1 This is a very popular issue (currently tied for 2nd place in votes). Unfortunately, I think the single-pass collector attached here doesn't scale very well to large maxDoc and/or large number of unique groups. Also, it pulls a DocTermsIndex on the top-level reader (costly in an NRT/reopen setting since it's not per-segment). So I decided to factor out parts of Solr's current two-pass approach into a shared "grouping" module. The downside of the two-pass approach is you run the query twice, automatically halving your QPS. (It's even worse because the grouping itself is somewhat computing intensive too). To try to help mitigate this, I also added a new CachingCollector, which just holds hits (docID and optionally score) up to a max allowed RAM consumption, and can then replay them for the 2nd pass. In includes a "max RAM" setting so that if too many hits are found, it stops caching (and you must then re-execute the query). But one nice side effect of the two-phased approach is that sharding is in theory straightforward (I think?). Ie, all shards would do the first phase, concurrently, to get the top N groups. Then you merge-sort the resulting top groups, then run second phase (finding docs w/in the top groups) on all shards, then merge results from the same group across all shards.
3. Initial rough patch. I think it's working well. I reused the test case from the original patch (thank you Martijn!), and also created a new random test case. There are still nocommits/sops/javadocs to clean up...

but I think it's close. We cannot yet cutover Solr to this module because it doesn't support grouping by ValueSource (from a function query) nor by arbitrary query. (This refactoring groups by a single-valued indexed field.) I think we need to first refactor function queries (LUCENE-2883) and also filter caches out of Solr before cutting Solr over. I plan to backport this to 3.x as contrib/grouping.

4. The issue I have with the group=true feature in Solr is that the facets are not calculated post grouping. So I cannot show the (count) in the facet list for a field. If we can get the facets to return counts POST grouping that would be ideal. Bill
5. Nice work Michael! I also think that the two pass mechanism is definitely the preferred way to go. I think we also need a strategy mechanism (or at least an GroupCollector class hierarchy) inside this module. The mechanism should select the right group collector(s) for a certain request. Some users maybe only care about the top group document, so I second pass won't be necessary. Another example with faceting in mind. When group based faceting is necessary. The top N groups don't suffice. You'll need all group docs (I currently don't see a other way). These groups docs are then used to create a grouped Solr DocSet. But this should be a completely different implementation.
6. Patch w/ next iteration... I beefed up the overview.html, added test case coverage of "null" groupValue. I think it's ready to commit and then back-port to 3.x!
7. {quote} I think we also need a strategy mechanism (or at least an GroupCollector class hierarchy) inside this module. The mechanism should select the right group collector(s) for a certain request. Some users maybe only care about the top group document, so I second pass won't be necessary. Another example with faceting in mind. When group based faceting is necessary. The top N groups don't suffice. You'll need all group docs (I currently don't see a other way). These groups docs are then used to create a grouped Solr DocSet. But this should be a completely different implementation. {quote} I agree, there's much more we could do here! Specialized collection for the maxDocsPerGroup=1 case, and for the "I want all groups" case, would be nice. For the "not many unique values in the group field" case we could do a single-pass collector, I think. Grouping by a multi-valued field should be possible (we now have DocTermOrds in Lucene, but it doesn't load the term byte[] data), as well as support for sharding, ie, by merging top groups and docs w/in each group (but I think we need an addition to FieldComparator API for this). I think we should commit this starting point, today, and then iterate from there... Martijn, thank you for persisting for so long on SOLR-236! We are finally getting grouping functionality accessible from Lucene and Solr...
8. bq. If we can get the facets to return counts POST grouping that would be ideal. How would the field values for the group be defined...? Or would facets run on all not-collapsed docs...?
9. Say we have 4 documents: docid=1 hgid=1 age=10 docid=2 hgid=1 age=10 docid=3 hgid=2 age=12 docid=4 hgid=4 age=11 If we group by hgid, we would get: hgid=1 >docid=1 hgid=1 age=10 >docid=2 hgid=1 age=10 hgid=3 >docid=3 hgid=2 age=12 hgid=4 >docid=4 hgid=4 age=11 If I set Facet Counts = POST age: 10 (1 document) age: 11 (1 document) age: 12 (1 document) If I set Facet Counts = PRE age: 10 (2 document) age: 11 (1 document) age: 12 (1 document) The only way grouping works in Solr now is Facet Counts = PRE. Thanks.
10. But what if docid=2 had age=17 instead? How would we determine what value the group (for hgid=1) should have for the "age" field? Or... would the group count +1 to age=10 and +1 to age=17 in that case? (ie, as if the group were a single document w/ multi-valued field age).
11. bq. But what if docid=2 had age=17 instead? How would we determine what value the group (for hgid=1) should have for the "age" field? That would depend on the group sort, right? If your group sort is age asc the lowest document would be chosen. bq. Or... would the group count +1 to age=10 and +1 to age=17 in that case? You mean like a sum of all ages per group? That is interesting, but sounds more like a function to me. This can be computed with a separated group collector. Wouldn't make sense to me, to have this with a regular field facet.
12. {quote} bq. But what if docid=2 had age=17 instead? How would we determine what value the group (for hgid=1) should have for the "age" field? That would depend on the group sort, right? If your group sort is age asc the lowest document would be chosen. {quote} OK, I see. So the group is "represented" by the doc within it that sorts highest according to the group sort, and any faceting on the groups means faceting on that top doc's values, per group. Neat. {quote} bq. Or... would the group count +1 to age=10 and +1 to age=17 in that case? You mean like a sum of all ages per group? That is interesting, but sounds more like a function to me. This can be computed with a separated group collector. Wouldn't make sense to me, to have this with a regular field facet. {quote} Well, not sum, but multi-valued? (Ie, as if this group were represented by a doc that takes the union of all age values of docs within it). This way, if the user then does a drill-down by a specific age, the number of groups then returned would match the facet count of that age in the first query. I agree we need to hash out these semantics :) Bill could you open a separate

Lucene issue, to work out the semantics & impl for "post-grouping-faceting"? Unfortunately, it's blocked by factoring out the facet module (LUCENE-3079).

13. Michael I see you have committed it to the trunk. Nice work! Only one quest why is the SearchGroup class now package protected? For me the documentation in overview.html suggest that I can just use it in any package. As for porting this code to the 3x branch I see that this branch doesn't have modules. Does it mean that it will be a Lucene contrib?
14. Woops, it should not be protected! I'll fix... thanks Martijn! Yes, I ported to 3.x as contrib/grouping, so this will be released when we release 3.2.
15. Hi Martijn, in 3.x it is available as a lucene contrib (contrib/grouping). As far as classes being package-private, I don't think Mike intended this, as they are marked experimental. Want to upload a patch that ensures everything you need has correct visibility?
16. bq. Want to upload a patch that ensures everything you need has correct visibility? Only the SearchGroup class had different visibility compared to the patch. And Michael says he is going to change that. So I think a patch for that is a bit overkill. bq. Yes, I ported to 3.x as contrib/grouping, so this will be released when we release 3.2. Cool! After a svn update I see the contrib now as well. The question is how to go from here. Continue development in this issue or for each new grouping related feature a separate issue? E.g. think about class hierarchy / strategy for grouping collectors. And new collectors like UniqueFieldValueCountGroupCollector. I haven't see an issue regarding post faceting yet. I can create one if necessary?
17. **body:** {quote} Only the SearchGroup class had different visibility compared to the patch. And Michael says he is going to change that. So I think a patch for that is a bit overkill. {quote} Ok, thanks for taking a look... I just figured we could tackle any visibility issues at once, but it seems this is the only one. {quote} I haven't see an issue regarding post faceting yet. I can create one if necessary? {quote} Sure! of course it will unfortunately need to be blocked on LUCENE-3079, but it seems like a good idea to have the issue open for planning. {quote} The question is how to go from here. Continue development in this issue or for each new grouping related feature a separate issue? E.g. think about class hierarchy / strategy for grouping collectors. And new collectors like UniqueFieldValueCountGroupCollector. {quote} My opinion would be to open new issues for each new grouping feature! This way things can get committed faster and its easier to review patches (when there are enormous jira issues with a mix of unrelated patches its pretty difficult to keep up with what is happening). These apis are marked experimental so I don't think we should waste time on backwards compatibility, nor should we try to come up with "holy grail" solutions that solve everyones problems in one single commit... I think we should go iteratively. Patches welcome!
    label: code-design
18. Resolving this... we can iterate in further issues. Thanks Martijn!
19. **body:** I adding grouping queries to the nightly benchmarks (http://people.apache.org/~mikemccand/lucenebench) -- see TermGroup100/10K/1M. The "F" annotation is the day grouping queries first ran. Those queries are the same queries running as TermQuery, just with grouping turned on on 3 randomly generated fields, with 100, 10,000 and 1 million unique values. So we can gauge the perf hit by comparing to TermQuery each night. I use the CachingCollector. First off, I'm impressed that the perf hit for grouping is not too bad: ||Query||QPS||Slowdown|| |TermQuery (baseline)|30.72|0| |TermGroup100|13.59|2.26| |TermQuery10K|13.2|2.34| |TermQuery1M|12.15|2.53| I had expected we'd pay a bigger perf hit! Second, there more unique groups you have, the slower grouping gets, but that multiplier really isn't so bad -- the 1M unique groups case is only 10.6% slower than the 100 unique groups case. Remember, though, that these groups are randomly generated full-unicode strings, so real data could very well produce different results... Third, and this is insanity, the addition of grouping caused other unexpected changes. Most horribly, SpanNearQuery slowed down by ~12.2% (http://people.apache.org/~mikemccand/lucenebench/SpanNear.html), while other queries seem to get a bit faster. I think this is [frustratingly!] due to hotspot making different decisions about which code to optimize/inline. Similarly strange, when I added sorting (TermQuery sorting by title and date/time, "E" annotation in all graphs), I saw the variance in the unsorted TermQuery performance drop substantially. I'm pretty sure this wide variance was due to hotspot's erratic decision making, but somehow the addition of sorting, while not change TermQuery's mean QPS, caused hotspot to at least be somewhat more consistent in how it compiled the code. Maybe as we add more and more diverse queries to the benchmark we'll see hotspot behave more "reasonably"....
    label: code-design
20. bq. I adding grouping queries to the nightly benchmarks Nice! Are the regular sort and group sort different in these test cases? Do think when new features are added that these also need be added to this test suite? Or is this perfomance test suite just for the basic features?

21. **body:** I'm only testing groupSort and sort by relevance now in the nightly bench. I'll add sort-by-title, groupSort-by-relevance cases too, so we test that. Hmm, though: this content set is alphabetized by title I believe, so it's not really a good test. (I suspect that's why the TermQuery sorting by title is faster bq. Do think when new features are added that these also need be added to this test suite? Or is this perfomance test suite just for the basic features? Well, in general I'd love to have wider coverage in the nightly perf test... really it's only a start now. But there's no hard rule we have to add new functions into the nightly bench...
    label: test
22. Bulk closing for 3.2