Item 56
git_comments:

1. passing a negative number for FACET_THREADS implies an unlimited number of threads is acceptable. Also, a subtlety of directeExecutor is that no matter how many times you "submit" a job, it's really just a method call in that it's run by this thread.
2. Someone else has put the place holder in, wait for that to complete.
3. This thread will load this field, don't let other threads try.
4. Note, this cleverly replaces the placeholder.
5. Dummy for synchronization. cheapest initialization I can find.
6. Should at least return the placeholder, NPE if not is OK. OK, another thread put this in the cache we should be good.
7. All I really care about here is the chance to fire off a bunch of threads to the UnIninvertedField.get method to insure that we get into/out of the lock. Again, it's not entirely deterministic, but it might catch bad stuff occasionally...
8. Now, are all the UnInvertedFields still the same? Meaning they weren't re-fetched even when a bunch were requested at the same time?
9. Gimme 50 docs with 10 facet fields each
10. After this all, the uninverted fields should be exactly the same as they were the first time, even if we blast a whole bunch of identical fields at the facet code. Which, BTW, doesn't detect if you've asked for the same field more than once. The way fetching the uninverted field is written, all this is really testing is if the cache is working. It's NOT testing whether the pending/sleep is actually functioning, I had to do that by hand since I don't see how to make sure that uninverting the field multiple times actually happens to hit the wait state.
11. * * Numeric option indicating the maximum number of threads to be used * in counting facet field vales

git_commits:

1. **summary:** SOLR-2548, Multithread faceting
   message: SOLR-2548, Multithread faceting git-svn-id: https://svn.apache.org/repos/asf/lucene/dev/branches/branch_4x@1520670 13f79535-47bb-0310-9956-ffa450edef68

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Multithreaded faceting
   description: Add multithreading support for faceting.
2. **summary:** Multithreaded faceting
   description: Add multithreading support for faceting.
3. **summary:** Multithreaded faceting
   description: Add multithreading support for faceting.
   label: code-design
4. **summary:** Multithreaded faceting
   description: Add multithreading support for faceting.
   label: code-design
5. **summary:** Multithreaded faceting
   description: Add multithreading support for faceting.
6. **summary:** Multithreaded faceting
   description: Add multithreading support for faceting.

label: code-design

7. **summary:** Multithreaded faceting
   description: Add multithreading support for faceting.
8. **summary:** Multithreaded faceting
   description: Add multithreading support for faceting.
   label: code-design
9. **summary:** Multithreaded faceting
   description: Add multithreading support for faceting.
10. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
11. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
12. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
13. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
14. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
15. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
16. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
17. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
18. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
19. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
20. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
21. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
22. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
23. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
24. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
25. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
26. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
27. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: requirement
28. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
29. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
30. **summary:** Multithreaded faceting

description: Add multithreading support for faceting.

31. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
32. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
33. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
34. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
35. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
36. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
37. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
38. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
39. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
40. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: documentation
41. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
42. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
43. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
44. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
45. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
46. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
47. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
48. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
49. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
50. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
51. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
52. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
53. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
54. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
55. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design

56. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
57. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
58. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
59. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
60. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
61. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
    label: code-design
62. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.
63. **summary:** Multithreaded faceting
    description: Add multithreading support for faceting.

jira_issues_comments:

1. Patch for TRUNK
2. Patch for 3.1
3. **body:** The attached patch adds initial multithreading support for faceting. The patch simply wraps the facet counting methods per field into java.util.concurrent.Callable instances and executes these with a ExecutorService having a threadpool-size equal to the number of processors reported by the runtime. Seems like this adds quite nice speed boosts when faceting over multiple fields. In initial tests (with a patched SOLR 3.1 instance) the faceting speed was about 2x-8x faster with both faceting methods, enum and fc.
   label: code-design
4. **body:** Hello Janne, Is there any reason why you didn't make facet queries and facet ranges multi-threaded ? You create a thread pool with ${nProcessors} worker threads locally, meaning that a server with 8 processors serving 10 concurrent requests would have to start and then run 80 threads simultaneously before stopping each of them. At this level, multi-threaded facet computation may be slower than single-threaded facet computation. I suggest instead to share a single thread pool for every facet computation with coreSize=0, a SynchronousQueue, and ThreadPoolExecutor.CallerRunsPolicy as a RejectExecutionHandler. This way you would avoid the overhead of starting and then stopping threads and have a better control over the total number of threads computing facets.
   label: code-design
5. Hi Adrien, It's a initial patch. I don't use facet queries / ranges, so I started with field counts. What I'm planning to do is to run some stress tests against a 500M documents index distributed over 54 cores (once my index-building process is finished). I will definitely take a look at your suggestion once I'm able to run the stress tests against the 500M docs index. If the multi-threaded faceting version performs better under heavy load, I'll look into the possibility of making the facet queries + ranges multi-threaded.
6. **body:** Janne: thanks for the awesome patch! in general i think this type of functionality is a good idea -- the real question is how it should be configured/controlled. admins with many CPUs who expect low amounts of concurrent user traffic might be ok with spawning availableProcessors() threads per request, but admins with less CPUs then concurrent requests are going to prefer that individual requests stay single threaded and take a little longer. The suggestion to use a thread pool based executorservice definitely seems like it makes more sense, then it just becomes a matter of asking the admin to configure a simple number determining the size of the threadpool, we just need to support a few sentinal values: NUM_CPUS, and NONE (always use callers thread). since we'd want a threadpool that lives longer then a single request, this definitely shouldn't be an option specified via SolrParams (not to mention the risk involved if people don't lock it down with invariants). That leaves the question of wether this should be an "init" param on the FacetComponent, or something more global. My first thought was that we should bite the bullet and add a new top level config for a global thread pool executor service that any solr plugin could start using, but after thinking about it some more i think that would not only be premature, but perhaps even a bad idea in general -- even if we assume something like DIH, UIMA, or Highlighting

could also take advantage of a shared thread pool owned by solr, if you really care about parallelizing faceting, you probably wouldn't want some other intensive component starving out the thread pool (or vice versa)
label: code-design
7. I think this should be configurable on a per-request basis (not the max size of the threadpool, but how many threads out of that to use concurrently). For facet.method=fcs (per-segment faceting using the field cache), I did introduce a "threads" localParam. Perhaps we should have a "threads" or "facet.threads" request parameter?
8. **body:** Hoss Man, Regarding the best value of the number of threads to spawn based on the number of CPUs and the traffic, one could imagine to decide whether to spawn a new thread or to run the task in the current thread based on the load of the server. This way, servers under high traffic would run every request in a single thread (maximizing throughput) whereas servers under low traffic would be able to use every processor in order to minimize response time. The load is easily retrievable in Java 6 using OperatingSystemMXBean, I don't know if it is possible in a non OS-specific way in Java 5. I don't really understand what you mean by "if you really care about parallelizing faceting, you probably wouldn't want some other intensive component starving out the thread pool". Do you mean that you would expect some requests to be run slower with every component using a global thread pool than with a single thread pool dedicated to facets? Yonik, why would you want to limit the number of threads on a per-request basis, if enough CPUs are available?
label: code-design
9. my read on this patch is that there is no contention (not to say endorsement, but no contention) that faceting over multiple fields gets a speed boost. The outstanding questions seem to center on the configuration and how to expose and control the functionality. Given that two years has elapsed from the OP's original post - are some of the outstanding configuration/control questions now more easily resolved? In short, has SOLR moved forward such that answering these questions is now easier than it was the time of 3.1?
10. **body:** I would like to revive this ticket, if possible. We have an index with about 10 fields that we regularly facet on. These fields are either multi-valued or are of type TextField, so facet code chooses FC as the facet method, and uses the UnInvertedField instances to count each facet field, which takes several seconds per field in our case. So, multi-thread execution of getTermCounts() reduces the overall facet time considerably. I started with the patch that was posted against 3.1 and modified it a little bit to take into account previous comments made by Yonik and Adrien. The new patch applies against 4.2.1, uses the already existing facetExecutor thread pool, and is configured per request via a facet.threads request param. If the param is not supplied, the code defaults to directExecutor and runs sequential as before. So, code should behave as is if user chooses not to submit number of threads to use. Also in the process of testing, I noticed that UnInvertedField.getUnInvertedField() call was synchronized too early, before the call to new UnInvertedField(field, searcher) if the field is not in the field value cache. Because its init can take several seconds, synchronizing on the cache in that duration was effectively serializing the execution of the multiple threads. So, I modified it (albeit inelegantly) to synchronize later (in our case cache hit ratio is low, so this makes a difference). The patch is still incomplete, as it does not extend this framework to possibly other calls like ranges and dates, but it is a start.
label: code-design
11. Patch against 4.2.1
12. **body:** See Gun's comments about UnInvertedField serializing the facet counts due to the placement of the new viz. the synch block. It's at the very end of the class.... Do people think that the chance of uninverting the same field more than once and throwing away 2...N is frequent enough to guard against with a Future (or whatever?) It seems like this is an expensive enough operation that the complexity is reasonable.
label: code-design
13. This bit in SimpleFacets.getFacetFieldCounts bothers me: int maxThreads = Integer.parseInt(req.getParams().get(FacetParams.FACET_THREADS, "0")); Executor executor = maxThreads == 0 ? directExecutor : facetExecutor; maxThreads = maxThreads <= 0? Integer.MAX_VALUE : maxThreads; It seems like if the user doesn't specify anything for FACET_THREADS, they wind up spawning as many threads as there are facet fields specified. Probably not a real problem given this list will be fairly small, but it seems more true to the old default behavior if it's changed to something like int maxThreads = Integer.parseInt(req.getParams().get(FacetParams.FACET_THREADS, "1")); Executor executor = maxThreads == 1 ? directExecutor : facetExecutor; maxThreads = maxThreads < 1 ? Integer.MAX_VALUE : maxThreads; Or am I seeing things that aren't there?

14. **body:** Latest version that does two things: 1> does the max thread change I commented on earlier. 2> puts in some checking to insure that if multiple threads try to uninvert the same field at the same time, it'll only be loaded once. I used a simple wait/sleep loop here since this method is called from several places and it looks like a real pain to try to do a Future or whatever.
    label: code-design
15. **body:** bq. This bit in SimpleFacets.getFacetFieldCounts bothers me: ... bq. It seems like if the user doesn't specify anything for FACET_THREADS, they wind up spawning as many threads as there are facet fields specified I haven't reviewed the patch, but based on the snippet you posted i suspect you are reading that bit correctly. If FACET_THREADS isn't specified, or if it's specified and equals the default value of 0, then the directExecutor is used and _no_ threads should be spawned at all -- the value of maxThreads shouldn't matter at that point, instead the existing request thread should processes all of them sequentially. I'm guessing you should change the patch back. Side comments... 1) Something sketchy probably does happen if a user passes in a negative value -- it looks like that's the case when facetExecutor will be used with an unlimited number of threads ... that may actually have been intentional -- that if you say facet.threads=-1 every facet.field should get it's own thread, no matter how many there are, but if that is intentional i'd love to see a comment there making that obvious. (and a test showing that it works). 2) can you please fix that Integer.parseInt(..."0")) to just use params.getInt(...,0) ... that way the correct error message will be returned if it's not an int (and it's easier to read)
    label: code-design
16. [~hossman_lucene@fucit.org] Thanks. Your comments made me look more carefully at directExecutor, it took me a bit to wrap my head around that one. 1> Still checking on the implications of stacking up a bunch of directExecutors all through the CompletionService, not something I've used recently and the details are hazy. As far as tests are concerned, I haven't gotten there yet, the original patch didn't have any... It should be easy to create tests with multiple field.facet clauses, TestFaceting does this so there are templates. Is there a decent way to check whether more than one thread was actually spawned? If so, can you point me at some code that actually does that? Otherwise I'll create tests that just get the right response for single and multiple facet.field specifications and a bit of walk-through with the debugger to insure we actually go through that code path. 2> done. Thanks again.
17. bq. Still checking on the implications of stacking up a bunch of directExecutors all through the CompletionService, not something I've used recently and the details are hazy. unless i'm missing something, it should be a single directExecutor, and when a job is submitted to the CompletionService, nothing happens in the background at all -- the thread that submitted the job then immediately executes the job. Telling the COpmletionService to use the directExecutor is essentially a way of saying "when someone asks you to do execute X, make them do it themselves" bq. Is there a decent way to check whether more than one thread was actually spawned? I doubt it ... but it would be nice to at least know the functionality succeeds w/o failure. There might be a way to subclass & instrument the ThreadPoolExecutor (or the Queue it uses to manage jobs) so that you could make it keep track of the max number of live threads at any one time, or the max size of the queue at any one time, and then your test could reach in and inspect either of those values to know if the _wrong_ thing happened (ie: too many threads spun up, or too many things enqueued w/o being handed to threads) ... but i'm not sure how hard that would be. Acctually -- maybe a better thing to do would be to have the Callables record the thread id of whatever thread executed them, and include that in the debug info ... then the test could just confirm that all of the ids match and don't start with "facetExecutor-" in the directExecutor case, and that the number of unique ids seen is not greater then N in the facet.threads=N case. (That debug info could theoretically be useful to end users as well, to see that multiple threads really are getting used)
18. **body:** Hmm, the whole recording-thread-info is a little more ambitious than I want to be right now. For the nonce, I did some "by hand" debugging, added in a couple of (temporary) print message in the getUnInvertedField code and insured that when it's called it only executes once per field, so I think I'll call that good now. I did play around with the directExcecutor and now I get to add another bit of knowledge, that it's really kind of cool that it allows one to have code like this. No matter how many times you submit a job, it all just executes in the current thread. Arcane, but kind of cool. As for the rest, I've added at least functional tests and one test that the caching code is working that's non-deterministic but might trip bad conditions at least some of the time. So unless people object I'll be committing this probably tomorrow. It passes precommit and at least all the tests in TestFaceting, I'll be running the full suite in a minute.
    label: code-design
19. bq. I used a simple wait/sleep loop here Ugh - please let's not do that for multi-threaded code. Also, I see some stuff like this in the patch: {code} - counts = getGroupedCounts(searcher, docs, field, multiToken, offset,limit, mincount, missing, sort, prefix); + counts = getGroupedCounts(searcher, base, field,

multiToken, offset,limit, mincount, missing, sort, prefix); {code} Was there a bug that these changes fixed?

20. **body:** bq: Was there a bug that these changes fixed? Nope, I thought it was a refactoring and didn't look closely. It appears to be useless complexity, perhaps a remnant from the original patch against 3.1. I took them out. bq: please let's not do that for multi-threaded code. I can always count on you to call me on sleeping, don't know why I even try to put a sleep in any more :). OK, took it out and substituted a notifyAll. And added a test that gets into this code while actually doing the inverting rather than just pulls stuff from the cache. I'll attach a new patch in a few.
    label: code-design
21. bq. It appears to be useless complexity, perhaps a remnant from the original patch against 3.1. I took them out. Actually, I see now (and it's absolutely needed ;-) The base docset can change from one facet request to another (think excludes), hence if we go multi-threaded, we can't reference "SimpleFacets.docs" in anything that could be executed from a separate thread.
22. **body:** One issue with a static "pending" set on UnInvertedField is that it will block different cores trying to un-invert the same field. This should probably be implemented the same way the FieldCache does it (insertion of a placeholder).
    label: code-design
23. OK, maybe this time. 1> put back the passing in base. 2> took out the sleep. 3> changed how exceptions are propagated up past the new threads which fixed another test that this code broke. 4> Added a non-deterministic test that forces parallel uninverting of the fields to make sure we exercise the synchronize/notify code. This test can't _guarantee_ to execute that code every time, but it did manage with some printlns. Running tests again, precommit all that. Won't check in until at least tomorrow. And thank heaven for "local history" in IntelliJ ;)
24. **body:** Still have a test error in TestDistributedGrouping, no clue why and can't look right now. It's certainly a result of the changes in UnInvertField since if I put that in a clean trunk the same problem occurs. My guess is that I can't synchronize on cache for some reason, but not much in the way of evidence for that right now.
    label: code-design
25. **body:** [~yonik] Don't quite see what you're getting at. I understand what you're saying about pending blocking threads loading the same field name in different cores, good point. But how to put a placeholder in the cache? It needs an UnInverted field as an entry. I could add a bogus c'tor to make an degenerate UnInverted field and use _that_, then check every time we get something out of the cache for a field in order to see if it's the degenerate case. One could add a member var "imFake" or something. Really the question is how to distinguish between the cache returning null or returning something signaling "Come back later and get the entry another thread loaded". I like the idea of not having the spare pending set at all, one less thing to coordinate. Or I could just make the pending bits prepend the core name to the field? Actually, I'm beginning to wonder whether adding all this junk in is really better than just throwing the UnInvertedFields 2-n on the floor like the original patch did...
    label: code-design
26. See FieldCacheImpl.get() - there is a CreationPlaceHolder object used.
27. **body:** The latest patch isn't thread-safe in UnInvertedField - you're synchronizing on "cache" (of which there will be multiple) for accessing the singleton "pending".
    label: requirement
28. bq: you're synchronizing on "cache" Yeah, I realized that on the way in to an appointment. Siiigh.
29. bq: See FieldCacheImpl.get() - there is a CreationPlaceHolder object used. Right, but the map that that's placed _in_ is a map<key, Object>. The UnInvertedField cache is map<key, UnInvertedField>. That's what was behind my question about making a dummy UnInvertedField to use similarly to how CreationPlaceHolder is used. I'm really not up for making the underlying UnInvertedField cache take an Object, seems like the tail wagging the dog. Interestingly I think it was what [~yonik@apache.org] pointed out about synching on different objects than I thought I was that was the problem with TestDistributedGrouping, it passes now. The attached patch implements creating a placeholder UnInvertedField, removes the pending map and passes the failing test from yesterday as well as precommit. I'll run the full suite soon. If that passes, I'll let it stew for a bit and commit tomorrow unless there are objections.
30. Final patch, including CHANGES.txt entry.
31. Commit 1520645 from [~erickoerickson] in branch 'dev/trunk' [https://svn.apache.org/r1520645] SOLR-2548, Multithread faceting
32. Commit 1520670 from [~erickoerickson] in branch 'dev/branches/branch_4x' [https://svn.apache.org/r1520670] SOLR-2548, Multithread faceting

33. Thanks Janne and Gun!
34. **body:** This issue just got on my radar; I like working on threading problems. I commend the progress made but I think it can be improved: # I think it's counter-intuitive that if a user supplies facet.threads=2 then 3 cpu cores will actually be used (assuming >2 fields to facet on) # Only the first facet.threads worth of facets are actually done concurrently; the rest are done serially. # Even if the previous problem was solved, the use of the main calling thread to compute facets (beyond facet.threads) means that if by bad luck the main thread is computing the most intensive facets to compute, the other threads will sit idle once they are done when it would be better to have remaining work queued up. # in the event of an exception in one worker; the rest should be cancelled # ExecutionException is a wrapping exception; you should unwrap it and wrap SolrException on its contents, not the ExecutionException itself. The attached patch fixes all these problems, keeps it no more complex and perhaps simpler (IMO), and without increasing the lines-of-code count.
    label: code-design
35. Its a bad idea to call Future.cancel here. If any of the faceting methods are blocked on IO (e.g. docvalues faceting), this will close file descriptors with NIO/MMAP directory implementations: see the documentation in org.apache.lucene.store for more information.
36. [~dsmiley] See below... I'm not seeing points 1-3. I think you might be missing the distinction between adding fields to the pending queue and actually doing the faceting: (1) I don't think so. If facet.threads == 2, the third time around the counter is -1 so the field gets added to the pending structure, it's not executed on at all until one of the other threads completes. (2) I'm not seeing it. Every time a task completes, another is started from the pending list. The main thread is just sitting around waiting for the child threads to complete. Mostly this is for my edification, I have no objection to the semaphore approach. In fact it's a little cleaner, the second "for (String f : facetFs) {" loop is somewhat loosely coupled. (3) Not quite sure about this either. I don't see where the main thread is used to compute any facets. Well, except in the intentionally serial case when the directExecutor is used and the old behavior is desired. Items are just added to the pending queue once you exceed facet.threads. That queue is consumed to submit other tasks to new threads via "completionService.submit(pending.removeFirst());" in the second loop. The main thread never computes facets. Or I'm just blind to it. (4) That makes sense, although I'll defer to Robert. (5) OK. I did have some trouble in the tests though, some of them were expecting 400 response code and the SERVER_ERROR is 500 as I remember so don't be surprised if there's an issue there when you run the full test suite if you haven't already. I made some effort to give back the same errors as the tests expected which may account for some of the weirdness you saw in the exception handling. You'll notice I punted on Adrien's comment "Is there any reason why you didn't make facet queries and facet ranges multi-threaded"... feel free ;).
37. bq. Only the first facet.threads worth of facets are actually done concurrently; the rest are done serially. I remember that being my initial reaction too - but then when you think a little about it, you realize that it's not the case.
38. **body:** bq. in the event of an exception in one worker; the rest should be cancelled In addition to Robert's comment that points out why we never want to use cancel on anything that does IO, we shouldn't add complexity trying to optimize an error case. bq. ExecutionException is a wrapping exception; you should unwrap it and wrap SolrException on its contents, not the ExecutionException itself. We should definitely strive to make the multi-threading as transparent as possible (i.e. exceptions should be as close as possible to the non-threaded case).
    label: code-design
39. **body:** Just as a (likely controversial) suggestion in general here, its hard to "see" the single-threaded case (which is the most common case). I think its a little too sneaky here and would be actually a lot easier long-term if the single-threaded case was explicitly separate from the multi-threaded one.
    label: code-design
40. **body:** bq: Just as a (likely controversial) suggestion in general here, its hard to "see" the single-threaded case (which is the most common case). No, not controversial at all. I had to look at that pretty hard to see that it was a single-threaded case, I tried to add a comment, mostly so I wouldn't have to try to figure it out again next time I was in that code ;) I'm all in favor of a little more verbosity here, just didn't do it...
    label: documentation
41. **body:** bq. If any of the faceting methods are blocked on IO (e.g. docvalues faceting), this will close file descriptors with NIO/MMAP directory implementations: see the documentation in org.apache.lucene.store for more information. Ok; I'll look into that later. {quote} > Only the first facet.threads worth of facets are actually done concurrently; the rest are done serially. I remember that being my initial reaction too - but then when you think a little about it, you realize that it's not the case. {quote} Aha; now I see it! This is confusing code -- adding to the completionService/executor in two

different loops; and the 2nd loop is particularly un-obvious to me. bq. Just as a (likely controversial) suggestion in general here, its hard to "see" the single-threaded case (which is the most common case). +0 not controversial to me
label: code-design

42. **body:** BTW sorry for raising all these supposed shortfalls when the more serious ones have turned out to be invalid. I guess it just underscores what we all know -- multithreaded code is confusing. All the more reason to try to document it better and/or to try to code it clearly.
label: code-design
43. bq: sorry for raising all these supposed... NP. I would far, far rather have someone look at the code and raise any issues they see, even if they can be explained away than have code get into the wild and have to track it down later.... So far the structure of the code hasn't been obvious to anybody on first reading (you, Yonik and me first 2-3 times I looked at the patch). I probably won't remember it next week and would have to work laboriously through it again. It sure sounds like something that could stand some clarification/simplification.
44. **body:** The attached patch improves on my previous one a little -- a few more comments, a variable rename for clarity, an assertion. And of course I removed the future.cancel() loop. I think this code is pretty clear as far as multithreaded code goes: One loop that submits tasks, and a follow-on loop that consumes the results of those tasks, and a semaphore to ensure no more than the desired number of threads are computing the facets. It'd be cool to eventually extend multithreading across all the faceting types. I'll look into that next week.
label: code-design
45. So maybe just commit this when you think it's ready? I'll probably get a chance to look it over Tuesday on the airplane, but if you're happy with it feel free. We can always put the other faceting types into a new JIRA?
46. Commit 1523677 from [~dsmiley] in branch 'dev/trunk' [https://svn.apache.org/r1523677] SOLR-2548: Simplified multi-threading of facet.threads logic
47. I just committed to trunk; I'll wait a day just in case and for any more feedback before applying to 4x.
48. David: i'm not suggesting we rush this -- but if your changes aren't going to make it into 4.5, we should track them in a new issue that can have it's own record in CHANGES.txt so it's clear what versions of Solr have what version of the code.
49. I thought about that. I figure that if I'm cautious about this such as by committing to trunk first, as I did, then there shouldn't be consternation about porting this to branch_45. Besides, I have more confidence in understanding the code that I committed vs. what it replaced. But I take your point that *if* for some reason it doesn't go to v4.5 then, sure, use another issue.
50. Commit 1524066 from [~dsmiley] in branch 'dev/branches/branch_4x' [https://svn.apache.org/r1524066] SOLR-2548: Simplified multi-threading of facet.threads logic
51. Commit 1524080 from [~dsmiley] in branch 'dev/branches/lucene_solr_4_5' [https://svn.apache.org/r1524080] SOLR-2548: Simplified multi-threading of facet.threads logic
52. 4.5 release -> bulk close
53. Was just checking this out, very cool... One issue I see is that there is no way to limit the maximum number of threads specified at query time. This is configured statically in code to Integer.MAX_VALUE... this seems a bit scary to me.. especially when you don't have control over the types of queries being executed against the engine.
54. facet.threads is a new parameter to address exactly this concern, see: http://wiki.apache.org/solr/SimpleFacetParameters#facet.threads
55. **body:** When using facet.threads=1000 I am not seeing any better performance. 1. Does it work on facet.query as well as facet.field? 2. If I only have 1 facet.field - adding threads will it do anything? 3. Will it help more on multiple facet.field? 4. Does it help with facet.method=fc/fcs/ or enum?
label: code-design
56. 1. no. Could be extended to I think if you have the energy. 2. no 3. yes 4 all
57. **body:** I'm having a hard time measuring performance differenes without and with facet.threads. On my development machine, there are no differences on warmed indexes, both measure around 1ms. They're also almost identical after stop/start of Jetty with no warm up queries, around 40ms, after that, fast again. We're facetting on four fields this time, there are also four threads.
label: code-design
58. **body:** Alright, i took another index and facetted on much more fields and now i see a small improvement after start up of about 12%. It is not much, perhaps this machine is too fast in this case.
label: code-design

59. Multithreaded faceting is useful when your CPU core count is much greater than the number of Solr cores you have, and you have a ton of data and need to facet on multiple fields. You could theoretically get similar results by sharding more but you should limit sharding based on disk IO capabilities (especially when there's so much it won't get in RAM), which isn't necessary one-for-one with the CPU count.
60. We observed 4x speedup when calculating 14 facets in 6 threads for 200mln index. Thanks everybody! https://twitter.com/AlexKozhemiakin/status/389688204309196800
61. **body:** Alexey: Thanks for the feedback. This is one of those things that's very dependent on faceting on more then one field/query and the facets being fairly complex/expensive. But when those conditions are met, it's very noticeable.
    label: code-design