Item 81
git_comments:

git_commits:

1. **summary:** SOLR-8323: DocCollection.isFullyActive needs to know how many replicas to expect
   message: SOLR-8323: DocCollection.isFullyActive needs to know how many replicas to expect

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** Add CollectionWatcher API to ZkStateReader
   description: An API to watch for changes to collection state would be a generally useful thing, both internally and for client use.

jira_issues_comments:

1. **body:** Patch outlining the basic idea. This adds two new interfaces, CollectionStateWatcher and CollectionStatePredicate. The first can be registered for a particular collection with ZkStateReader and is called when the state of that collection changes (as determined by the internal watcher of that collection's state.json node). The second is used in a new ZkStateReader.waitForState() method, and is called on a state change to see if the state of a collection matches a predicate. There are also forwarding methods on CloudSolrClient for use by SolrJ clients, and a couple of helper methods on DocCollection and Replica to easily check for collection liveness. The new interfaces lend themselves nicely to use as Java 8 functional interfaces, and the TestCollectionStateWatchers test demonstrate both lambdas and method references here. This should make it easy to replace some of the helper methods (eg waitForThingsToLevelOut, waitForRecoveriesToFinish) in our tests with methods available to SolrJ. A caveat: this is only implemented for collections with their own state.json. I think it should be relatively easy to extend it to stateformat=1 collections as well if people think that's worth it.
   label: code-design
2. **body:** bq. waitForThingsToLevelOut That one is pretty test specific. bq. I think it should be relatively easy to extend it to stateformat=1 collections as well if people think that's worth it. Someone should remove stateformat=1 for Solr 6 in an ideal world.
   label: code-design
3. bq. Someone should remove stateformat=1 for Solr 6 in an ideal world Absolutely. Maybe this new API should just go into trunk for now? If Solr 6 is coming early in the new year it makes sense to start adding things that don't need to worry about back-compatibility.
4. Updated patch, using the SolrCloudTestCase from SOLR-8758. This has required a couple of tweaks to the collection-watching code in ZkStateReader, to allow for watching of non-existent collections.
5. **body:** - Why is DocCollection.isFullyActive() static? bq. stateWatchers.putIfAbsent(collection, Collections.synchronizedList(new ArrayList<>())); - You want computeIfAbsent() here to avoid the allocations. - If waitForState() exits with the TimeoutException, the watcher never gets removed. - There is a fundamental problem with how interestingCollections is getting managed now; there are external controls on that set, but now it's mixed up with the CollectionStateWatcher API. As an example, CollectionStateWatcher adds but never removes; and an external caller could call removeZkWatcher on a collection that there's a listener for. - The way the code is structured with setCreationWatch and refreshAndWatch doesn't make sense to me. Why in the heck are they recursive? I don't think you need all this. I suspect what you really want is to move the call to notifyStateWatchers() and handle it more intelligently to not fire events if the state hasn't actually changed. Basically, you want to call

notifyStateWatchers() from within updateWatchedCollection() exactly at the 3 points we're emitting log messages.
label: code-design

6. **body:** Thanks for the review, Scott! Here's an update patch. bq. Why is DocCollection.isFullyActive() static? Because the DocCollection passed to onStateChanged() may be null if the collection doesn't exist, or has been deleted. bq. If waitForState() exits with the TimeoutException, the watcher never gets removed. Fixed. bq. Basically, you want to call notifyStateWatchers() from within updateWatchedCollection() exactly at the 3 points we're emitting log messages Done, thanks - that's considerably simpler. bq. There is a fundamental problem with how interestingCollections is getting managed now I've restructured this entirely. Watches keep track of a) how many cores they have interested in them, and b) how many state watchers there are. Changes to a CollectionWatch state are always done inside a ConcurrentHashMap.compute() method to keep them atomic. This simplifies the watch handling in ZKController as well, and removes the abstraction leak where external objects controlled when to remove watches.
   label: code-design
7. I haven't forgotten this one, going to give the new patch a look this week (today or tomorrow)
8. Looking at this now. BTW, a Github PR might actually make this way easier....
9. I like the scheme of reference counting the ZkController core references
10. Could collectionWatches and interestingCollections be unified into a single thing? collectionWatches.keySet should always be equal to interestingCollections, so I don't a reason to have both
11. **body:** nit: make the static type of collectionWatchers be ConcurrentMap? Conveys intent better and plays nicer in IDE.
    label: code-design
12. unregisterCore needs a better guard against under-referencing, since it can be called from the outside. A caller could call unregisterCore enough times to make coreRefCount negative, offsetting a positive stateWatchers.size() and prematurely removing. Might even be advisable to throw an exception here on under reference.
13. **body:** {code} LOG.info("Deleting data for [{}]", coll); notifyStateWatchers(coll, newState); {code} newState is always null (IDE warning) so maybe just pass in null
    label: code-design
14. {code} if (watchers.size() == 0) return; {code} No need to early exit here, the loop will do it anyway
15. In notifyStateWatchers you can avoid some copies but just re-assigning the instance variable to a new empty set, and taking ownership of the existing set to fire events on.
16. In getStateWatchers() you probably still want to wrap in a compute function to avoid weird race conditions and memory-visibility problems. In particular there's absolutely no ordering guarantees on the reference to watch.stateWatchers
17. fetchCollectionState() expectExists parameter doesn't make sense to me... I would have thought that if a non-null watcher is passed in, you always want to setup an exists watch if the node doesn't exist. And if a null watcher is passed in, calling exists() is a waste of energy.
18. **body:** registerCore/ unregisterCore should probably retain the previous doc: /** This is not a public API. Only used by ZkController */
    label: documentation
19. getStateWatchers() could return null vs. empty set to differentiate between whether or not the collection is being watched, which would improve the precision of test assertions.
20. I did get one failure on a test run: "Did not see a fully active cluster after 30 seconds" But second run it passed.
21. **body:** Wow, thanks for the very thorough review Scott! Here's an updated patch. bq. Could collectionWatches and interestingCollections be unified into a single thing? Unfortunately not, as it's needed to detect collections which have migrated from state format 1 to state format 2. There's almost certainly a nicer way of doing that, though - maybe in a follow-up issue? bq. make the static type of collectionWatchers be ConcurrentMap? I disagree here - we don't use any of the concurrent methods, so I think just using Map is fine? bq. unregisterCore needs a better guard against under-referencing Added. I don't think throwing an exception is necessary, although maybe we should log a warning in this case? bq. newState is always null changed bq. No need to early exit here changed bq. In notifyStateWatchers you can avoid some copies... I think this ends up as a wash, given that you may end up creating multiple HashSets? And we're only copying references, after all. bq. In getStateWatchers() you probably still want to wrap in a compute function... Compute() doesn't help here, I don't think? And given that it's a test-only method, I'm not too concerned about accuracy. I've made it return a copy rather than return the original

set, which should stop weird things happening to it once it's been returned, though. bq. fetchCollectionState() expectExists parameter doesn't make sense to me Again, this is due to state format 1 - a collection state might be in clusterstate.json, so the collection-specific state might not exist. I agree about the null watcher though, and have added a check around the exists call for that. bq. registerCore/ unregisterCore should probably retain the previous doc: Added back bq. getStateWatchers() could return null vs. empty set Nice idea, added. I've also added an explicit test for state format 1 collections, and updated the code so that it actually works :)
label: code-design

22. Don't suppose I could convince you to open a github PR? Would make it much easier to review. :D
23. GitHub user romseygeek opened a pull request: https://github.com/apache/lucene-solr/pull/32 SOLR-8323 Adds a CollectionStateWatcher API to listen for changes to collection state (SOLR-8323) You can merge this pull request into a Git repository by running: $ git pull https://github.com/romseygeek/lucene-solr SOLR-8323 Alternatively you can review and apply these changes as the patch at: https://github.com/apache/lucene-solr/pull/32.patch To close this pull request, make a commit to your master/trunk branch with (at least) the following in the commit message: This closes #32 ---- ----
24. Pull request opened, review away! I see that you've already committed some changes to the way legacy collections are dealt with, so we may well be able to remove the 'interestingcollections' list - will give it a go.
25. **body:** [~dragonsinth] any comments on the pull request? I'd like to get this in soon, as it will make it easier to clean up a bunch of tests.
label: code-design
26. Sorry! I'll look at it today. Been swamped with other stuff. :(
27. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61503724 --- Diff: solr/solrj/src/java/org/apache/solr/client/solrj/impl/CloudSolrClient.java --- @@ -572,6 +574,40 @@ public void downloadConfig(String configName, Path downloadPath) throws IOExcept zkStateReader.getConfigManager().downloadConfigDir(configName, downloadPath); } + /** + * Block until a collection state matches a predicate, or a timeout + * + * Note that the predicate may be called again even after it has returned true, so + * implementors should avoid changing state within the predicate call itself. + * + * @param collection the collection to watch + * @param wait how long to wait + * @param unit the units of the wait parameter + * @param predicate a {@link CollectionStatePredicate} to check the collection state + * @throws InterruptedException on interrupt + * @throws TimeoutException on timeout + */ + public void waitForState(String collection, long wait, TimeUnit unit, CollectionStatePredicate predicate) + throws InterruptedException, TimeoutException { + connect(); + zkStateReader.waitForState(collection, wait, unit, predicate); + } + + /** + * Register a CollectionStateWatcher to be called when the cluster state for a collection changes + * + * Note that the watcher is unregistered after it has been called once. To make a watcher persistent, + * it should re-register itself in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * call + * + * @param collection the collection to watch + * @param watcher a watcher that will be called when the state changes + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher watcher) { + connect(); + zkStateReader.registerCollectionStateWatcher(collection, watcher); + } + --- End diff -- I would note that getZkStateReader() is a public method, is there value in adding these forwarding methods?
28. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61504017 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/CollectionStatePredicate.java --- @@ -0,0 +1,41 @@ +package org.apache.solr.common.cloud; + +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +import java.util.Set; +import java.util.concurrent.TimeUnit; + +/** + * Interface to determine if a collection state matches a required state + * + * @see ZkStateReader#waitForState(String, long, TimeUnit, CollectionStatePredicate) + */ +public interface CollectionStatePredicate { + + /** + * Check the collection state matches a required state + * + * The collectionState parameter may be null if the collection does not exist + * or has been

deleted --- End diff -- This wants to be `@param collectionState the current collection state, or null if the collection doesn't exist or has been deleted`

29. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61504670 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/CollectionStateWatcher.java --- @@ -0,0 +1,42 @@ +package org.apache.solr.common.cloud; + +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. +*/ + +import java.util.Set; + +/** + * Callback registered with {@link ZkStateReader#registerCollectionStateWatcher(String, CollectionStateWatcher)} + * and called whenever the collection state changes. + */ +public interface CollectionStateWatcher { + + /** + * Called when the collection we are registered against has a change of state + * + * Note that, due to the way Zookeeper watchers are implemented, a single call may be + * the result of several state changes + * + * A watcher is unregistered after it has been called once. To make a watcher persistent, + * implementors should re-register during this call. + * + * @param liveNodes the set of live nodes + * @param collectionState the new collection state + */ + void onStateChanged(Set<String> liveNodes, DocCollection collectionState); + +} --- End diff -- I just want to toss out an idea here after looking at this some more. I notice that CollectionStateWatcher and CollectionStatePredicate are nearly identical. What would you think about combining the two into a single interface? The signature could be e.g.: bool stateChanged(liveNodes, collectionState) In a watch context, the return value means "keep watching?". So return true to reset the watcher and continue getting updates, or return false to stop watching. In a predicate context, the return value means "keep waiting?". So return true to keep waiting, or return false if you've finally seen what you were waiting for. They'll both have the same semantic meaning either way.

30. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61504824 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/DocCollection.java --- @@ -210,6 +213,38 @@ public Replica getReplica(String coreNodeName) { return null; } + /** + * Check that all replicas in a collection are live + * + * @see CollectionStatePredicate + */ + public static boolean isFullyActive(Set<String> liveNodes, DocCollection collectionState) { + Objects.requireNonNull(liveNodes); + if (collectionState == null) + return false; + for (Slice slice : collectionState) { + for (Replica replica : slice) { + if (replica.isActive(liveNodes) == false) + return false; + } + } + return true; + } + + /** + * Returns true if the passed in DocCollection is null + * + * @see CollectionStatePredicate + */ + public static boolean isDeleted(Set<String> liveNodes, DocCollection collectionState) { + return collectionState == null; + } --- End diff -- maybe `exists`? isDeleted implies that it used to exist, but it may have never been created

31. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61507382 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -484,6 +498,12 @@ private void refreshLegacyClusterState(Watcher watcher) } this.legacyCollectionStates = loadedData.getCollectionStates(); this.legacyClusterStateVersion = stat.getVersion(); + for (Map.Entry<String, ClusterState.CollectionRef> entry : this.legacyCollectionStates.entrySet()) { + if (entry.getValue().isLazilyLoaded() == false) { + // a watched collection - trigger notifications + notifyStateWatchers(entry.getKey(), entry.getValue().get()); + } + } --- End diff -- I think it would add a lot of value here to actually check differences. There really wouldn't be much computational work since you could restrict it to watched collections. Something like: ``` for (Map.Entry<String, CollectionWatch> watchEntry : this.collectionWatches.entrySet()) { String coll = watchEntry.getKey(); CollectionWatch collWatch = watchEntry.getValue(); DocCollection newState = this.legacyCollectionStates.get(coll).get(); if (!collWatch.stateWatchers.isEmpty() && !Objects.equals(oldCollectionStates.get(coll).get(), newState)) { notifyStateWatchers(coll, newState); } } ```

32. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61507961 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -131,6 +132,19 @@ private final Runnable securityNodeListener; + private Map<String, CollectionWatch> collectionWatches = new ConcurrentHashMap<>(); --- End diff -- The reason I made a comment about using the concrete type here is that it makes it much easier to work with as a developer. When you can see that the static type

of this variable is ConcurrentHashMap, that helps you evaluate the code that touches it. For example, when you use IDE features to 'click through' a method call or view the javadoc on a called method, you get the ConcurrentHashMap version of the javadoc/method instead of the Map version, which helps you more easily evaluate the correctness.

33. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61508998 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -1066,32 +1079,201 @@ public static String getCollectionPath(String coll) { return COLLECTIONS_ZKNODE+"/"+coll + "/state.json"; } - public void addCollectionWatch(String coll) { - if (interestingCollections.add(coll)) { - LOG.info("addZkWatch [{}]", coll); - new StateWatcher(coll).refreshAndWatch(false); + /** + * Notify this reader that a local Core is a member of a collection, and so that collection + * state should be watched. + * + * Not a public API. This method should only be called from ZkController. + * + * The number of cores per-collection is tracked, and adding multiple cores from the same + * collection does not increase the number of watches. + * + * @param collection the collection that the core is a member of + * + * @see ZkStateReader#unregisterCore(String) + */ + public void registerCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + reconstructState.set(true); + v = new CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new StateWatcher(collection).refreshAndWatch(); + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Notify this reader that a local core that is a member of a collection has been closed. + * + * Not a public API. This method should only be called from ZkController. + * + * If no cores are registered for a collection, and there are no {@link CollectionStateWatcher}s + * for that collection either, the collection watch will be removed. + * + * @param collection the collection that the core belongs to + */ + public void unregisterCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) + return null; + if (v.coreRefCount > 0) + v.coreRefCount--; + if (v.canBeRemoved()) { + watchedCollectionStates.remove(collection); + lazyCollectionStates.put(collection, new LazyCollectionRef(collection)); + reconstructState.set(true); + return null; + } + return v; + }); + if (reconstructState.get()) { + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Register a CollectionStateWatcher to be called when the state of a collection changes + * + * A given CollectionStateWatcher will be only called once. If you want to have a persistent watcher, + * it should register itself again in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * method. + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher stateWatcher) { + AtomicBoolean watchSet = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + v = new CollectionWatch(); + watchSet.set(true); + } + v.stateWatchers.add(stateWatcher); + return v; + }); + if (watchSet.get()) { + new StateWatcher(collection).refreshAndWatch(); synchronized (getUpdateLock()) { constructState(); } } } + /** + * Block until a CollectionStatePredicate returns true, or the wait times out + * + * Note that the predicate may be called again even after it has returned true, so + * implementors should avoid changing state within the predicate call itself. + * + * @param collection the collection to watch + * @param wait how long to wait + * @param unit the units of the wait parameter + * @param predicate the predicate to call on state changes + * @throws InterruptedException on interrupt + * @throws TimeoutException on timeout + */ + public void waitForState(final String collection, long wait, TimeUnit unit, CollectionStatePredicate predicate) --- End diff -- @shalinmangar this is what I was referring to, I know you're working on getting Overseer to not peg ZK polling for state changes on unwatched collections, this PR provides an easy mechanism to temporarily watch collections of interest.
34. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61509699 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -1066,32 +1079,201 @@ public static String getCollectionPath(String coll) { return COLLECTIONS_ZKNODE+"/"+coll + "/state.json"; } - public void addCollectionWatch(String coll) { - if (interestingCollections.add(coll)) { - LOG.info("addZkWatch [{}]", coll); - new StateWatcher(coll).refreshAndWatch(false); + /** + * Notify this reader that a local Core is a member of a collection, and so that collection + * state should be watched. + * + * Not a public API. This method should only be called from ZkController. + * + * The number of cores per-collection is tracked, and adding multiple cores from the same + * collection does not increase the number of watches. + * + * @param collection the collection that the core is a member of + * + * @see ZkStateReader#unregisterCore(String) + */ + public void registerCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + reconstructState.set(true); + v = new CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new StateWatcher(collection).refreshAndWatch(); + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Notify this reader that a local core that is a member of a collection has been closed. + * + * Not a public API. This method should only be called from ZkController. + * + * If no cores are registered for a collection, and there are no {@link CollectionStateWatcher}s + * for that collection either, the collection watch will be removed. + * + * @param collection the collection that the core belongs to + */ + public void unregisterCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) + return null; + if (v.coreRefCount > 0) + v.coreRefCount--; + if (v.canBeRemoved()) { + watchedCollectionStates.remove(collection); + lazyCollectionStates.put(collection, new LazyCollectionRef(collection)); + reconstructState.set(true); + return null; + } + return v; + }); + if (reconstructState.get()) { + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Register a CollectionStateWatcher to be called when the state of a collection changes + * + * A given CollectionStateWatcher will be only called once. If you want to have a persistent watcher, + * it should register itself again in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * method. + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher stateWatcher) { + AtomicBoolean watchSet = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + v = new CollectionWatch(); + watchSet.set(true); + } + v.stateWatchers.add(stateWatcher); + return v; + }); + if (watchSet.get()) { + new StateWatcher(collection).refreshAndWatch(); synchronized (getUpdateLock()) { constructState(); } } } + /** + * Block until a CollectionStatePredicate returns true, or the wait times out + * + * Note that the predicate may be called again even after it has returned true, so + * implementors should avoid changing state within the predicate call itself. + * + * @param collection the collection to watch + * @param wait how long to wait + * @param unit the units of the wait parameter + * @param predicate the predicate to call on state changes + * @throws InterruptedException on interrupt + * @throws TimeoutException on timeout + */ + public void waitForState(final String collection, long wait, TimeUnit unit, CollectionStatePredicate predicate) --- End diff -- @shalinmangar this is what I was referring to, I know you're working on getting Overseer to not peg ZK polling for state changes on unwatched collections, this PR provides an easy mechanism to temporarily watch collections of interest.

CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new StateWatcher(collection).refreshAndWatch(); + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Notify this reader that a local core that is a member of a collection has been closed. + * + * Not a public API. This method should only be called from ZkController. + * + * If no cores are registered for a collection, and there are no {@link CollectionStateWatcher}s + * for that collection either, the collection watch will be removed. + * + * @param collection the collection that the core belongs to + */ + public void unregisterCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) + return null; + if (v.coreRefCount > 0) + v.coreRefCount--; + if (v.canBeRemoved()) { + watchedCollectionStates.remove(collection); + lazyCollectionStates.put(collection, new LazyCollectionRef(collection)); + reconstructState.set(true); + return null; + } + return v; + }); + if (reconstructState.get()) { + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Register a CollectionStateWatcher to be called when the state of a collection changes + * + * A given CollectionStateWatcher will be only called once. If you want to have a persistent watcher, + * it should register itself again in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * method. + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher stateWatcher) { + AtomicBoolean watchSet = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + v = new CollectionWatch(); + watchSet.set(true); + } + v.stateWatchers.add(stateWatcher); + return v; + }); + if (watchSet.get()) { + new StateWatcher(collection).refreshAndWatch(); synchronized (getUpdateLock()) { constructState(); } } } + /** + * Block until a CollectionStatePredicate returns true, or the wait times out + * + * Note that the predicate may be called again even after it has returned true, so + * implementors should avoid changing state within the predicate call itself. --- End diff -- I think we could tighten this code up to ensure that predicate never gets call concurrently from two different threads at the same time, this would simplify things for clients and handle the case of calling it twice when it succeeds immediately.

35. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61509937 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -1066,32 +1079,201 @@ public static String getCollectionPath(String coll) { return COLLECTIONS_ZKNODE+"/"+coll + "/state.json"; } - public void addCollectionWatch(String coll) { - if (interestingCollections.add(coll)) { - LOG.info("addZkWatch [{}]", coll); - new StateWatcher(coll).refreshAndWatch(false); + /** + * Notify this reader that a local Core is a member of a collection, and so that collection + * state should be watched. + * + * Not a public API. This method should only be called from ZkController. + * + * The number of cores per-collection is tracked, and adding multiple cores from the same + * collection does not increase the number of watches. + * + * @param collection the collection that the core is a member of + * + * @see ZkStateReader#unregisterCore(String) + */ + public void registerCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + reconstructState.set(true); + v = new CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new StateWatcher(collection).refreshAndWatch(); + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Notify this reader that a local core that is a member of a collection has been closed. + * + * Not a public API. This method should only be called from ZkController. + * + * If no cores are registered for a collection, and there are no {@link CollectionStateWatcher}s + * for that collection either, the collection watch will be removed. + * + * @param collection the collection that the core belongs to + */ + public void unregisterCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) + return null; + if (v.coreRefCount > 0) + v.coreRefCount--; + if (v.canBeRemoved()) { + watchedCollectionStates.remove(collection); + lazyCollectionStates.put(collection, new LazyCollectionRef(collection)); + reconstructState.set(true); + return null; + } + return v; + }); + if (reconstructState.get()) { + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Register a CollectionStateWatcher to be called when the state of a collection changes + * + * A given CollectionStateWatcher will be only called once. If you want to have a persistent watcher, + * it should register itself again in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * method. + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher stateWatcher) { + AtomicBoolean watchSet = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + v = new CollectionWatch(); + watchSet.set(true); + } + v.stateWatchers.add(stateWatcher); + return v; + }); + if (watchSet.get()) { + new StateWatcher(collection).refreshAndWatch(); synchronized (getUpdateLock()) { constructState(); } } } + /** + * Block until a CollectionStatePredicate returns true, or the wait times out + * + * Note that the

predicate may be called again even after it has returned true, so + * implementors should avoid changing state within the predicate call itself. + * + * @param collection the collection to watch + * @param wait how long to wait + * @param unit the units of the wait parameter + * @param predicate the predicate to call on state changes + * @throws InterruptedException on interrupt + * @throws TimeoutException on timeout + */ + public void waitForState(final String collection, long wait, TimeUnit unit, CollectionStatePredicate predicate) + throws InterruptedException, TimeoutException { + + final CountDownLatch latch = new CountDownLatch(1); + + CollectionStateWatcher watcher = new CollectionStateWatcher() { + @Override + public void onStateChanged(Set<String> liveNodes, DocCollection collectionState) { + if (predicate.matches(liveNodes, collectionState)) { + latch.countDown(); + } else { + registerCollectionStateWatcher(collection, this); + } + } + }; + registerCollectionStateWatcher(collection, watcher); + + try { + // check the current state + DocCollection dc = clusterState.getCollectionOrNull(collection); + if (predicate.matches(liveNodes, dc)) + return; + + // wait for the watcher predicate to return true, or time out + if (!latch.await(wait, unit)) + throw new TimeoutException(); + + } + finally { + removeCollectionStateWatcher(collection, watcher); + } + } + + /** + * Remove a watcher from a collection's watch list. + * + * This allows Zookeeper watches to be removed if there is no interest in the + * collection. + * + * @param collection the collection + * @param watcher the watcher + */ + public void removeCollectionStateWatcher(String collection, CollectionStateWatcher watcher) { + collectionWatches.compute(collection, (k, v) -> { + if (v == null) + return null; + v.stateWatchers.remove(watcher); + if (v.canBeRemoved()) + return null; + return v; + }); + } + + private void notifyStateWatchers(String collection, DocCollection collectionState) { --- End diff -- I think we should pass in liveNodes; particularly in cases where we're firing a bunch of watchers, or even firing watchers on a bunch of collections at once, we can avoid the repeated volatile reads.

36. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61510100 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/CollectionStateWatcher.java --- @@ -0,0 +1,42 @@ +package org.apache.solr.common.cloud; + +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. +*/ + +import java.util.Set; + +/** + * Callback registered with {@link ZkStateReader#registerCollectionStateWatcher(String, CollectionStateWatcher)} + * and called whenever the collection state changes. + */ --- End diff -- If we're not going to be firing events on all watchers whenever live_nodes changes, we should be very clear about this.
37. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61510208 --- Diff: solr/test-framework/src/java/org/apache/solr/cloud/MiniSolrCloudCluster.java --- @@ -348,7 +358,13 @@ public JettySolrRunner stopJettySolrRunner(int index) throws Exception { return jetty; } - protected JettySolrRunner startJettySolrRunner(JettySolrRunner jetty) throws Exception { + /** + * Add a previously stopped node back to the cluster + * @param jetty a {@link JettySolrRunner} previously returned by {@link #stopJettySolrRunner(int)} + * @return the started node + * @throws Exception on error + */ + public JettySolrRunner startJettySolrRunner(JettySolrRunner jetty) throws Exception { --- End diff -- Are the changes in this file related to this PR?
38. Github user dragonsinth commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-215576790 Looking good, a little more high-level feedback. @shalinmangar I think you should take a look also. Have you run the tests extensively? The first time I ran I got a failure, but after that it's been fairly reliable, but I haven't beasted.
39. Github user romseygeek commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61541687 --- Diff: solr/solrj/src/java/org/apache/solr/client/solrj/impl/CloudSolrClient.java --- @@ -572,6 +574,40 @@ public void downloadConfig(String configName, Path downloadPath) throws IOExcept zkStateReader.getConfigManager().downloadConfigDir(configName, downloadPath); } + /** + * Block until a collection state matches a predicate, or a timeout + * + * Note that the predicate may be called again even after it has returned true, so + * implementors should avoid changing state within the predicate call itself. + * + * @param collection the collection to watch + * @param wait how long to wait + * @param unit the units of the wait parameter + * @param predicate a {@link CollectionStatePredicate} to

check the collection state + * @throws InterruptedException on interrupt + * @throws TimeoutException on timeout + */ + public void waitForState(String collection, long wait, TimeUnit unit, CollectionStatePredicate predicate) + throws InterruptedException, TimeoutException { + connect(); + zkStateReader.waitForState(collection, wait, unit, predicate); + } + + /** + * Register a CollectionStateWatcher to be called when the cluster state for a collection changes + * + * Note that the watcher is unregistered after it has been called once. To make a watcher persistent, + * it should re-register itself in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * call + * + * @param collection the collection to watch + * @param watcher a watcher that will be called when the state changes + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher watcher) { + connect(); + zkStateReader.registerCollectionStateWatcher(collection, watcher); + } + --- End diff -- TBH, I don't think getZkStateReader() *should* be a public method - client code that's using CloudSolrClient shouldn't need to interact with low-level stuff like the state reader. Maybe we only need to expose waitForState() here though.

40. Github user romseygeek commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61541928 --- Diff: solr/test-framework/src/java/org/apache/solr/cloud/MiniSolrCloudCluster.java --- @@ -348,7 +358,13 @@ public JettySolrRunner stopJettySolrRunner(int index) throws Exception { return jetty; } - protected JettySolrRunner startJettySolrRunner(JettySolrRunner jetty) throws Exception { + /** + * Add a previously stopped node back to the cluster + * @param jetty a {@link JettySolrRunner} previously returned by {@link #stopJettySolrRunner(int)} + * @return the started node + * @throws Exception on error + */ + public JettySolrRunner startJettySolrRunner(JettySolrRunner jetty) throws Exception { --- End diff -- Yes, they're needed to ensure that the cluster is in the same state for all tests.
41. Github user romseygeek commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-215646194 Thanks for the comments! I'll try and incorporate your suggestions and see how far we get.
42. Github user romseygeek commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61568081 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/CollectionStateWatcher.java --- @@ -0,0 +1,42 @@ +package org.apache.solr.common.cloud; + +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. +*/ + +import java.util.Set; + +/** + * Callback registered with {@link ZkStateReader#registerCollectionStateWatcher(String, CollectionStateWatcher)} + * and called whenever the collection state changes. + */ --- End diff -- Hm, that's a good point. Maybe we should be doing just that?
43. Github user romseygeek commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61569303 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -1066,32 +1079,201 @@ public static String getCollectionPath(String coll) { return COLLECTIONS_ZKNODE+"/"+coll + "/state.json"; } - public void addCollectionWatch(String coll) { - if (interestingCollections.add(coll)) { - LOG.info("addZkWatch [{}]", coll); - new StateWatcher(coll).refreshAndWatch(false); + /** + * Notify this reader that a local Core is a member of a collection, and so that collection + * state should be watched. + * + * Not a public API. This method should only be called from ZkController. + * + * The number of cores per-collection is tracked, and adding multiple cores from the same + * collection does not increase the number of watches. + * + * @param collection the collection that the core is a member of + * + * @see ZkStateReader#unregisterCore(String) + */ + public void registerCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + reconstructState.set(true); + v = new CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new StateWatcher(collection).refreshAndWatch(); + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Notify this reader that a local core that is a member of a collection has been closed. + * + * Not a public API. This method should only be called from ZkController. + * + * If no cores are registered for a collection, and there are no {@link CollectionStateWatcher}s + * for that collection either, the collection watch will be removed. + * + * @param collection the collection that the core

belongs to + */ + public void unregisterCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) + return null; + if (v.coreRefCount > 0) + v.coreRefCount--; + if (v.canBeRemoved()) { + watchedCollectionStates.remove(collection); + lazyCollectionStates.put(collection, new LazyCollectionRef(collection)); + reconstructState.set(true); + return null; + } + return v; + }); + if (reconstructState.get()) { + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Register a CollectionStateWatcher to be called when the state of a collection changes + * + * A given CollectionStateWatcher will be only called once. If you want to have a persistent watcher, + * it should register itself again in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * method. + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher stateWatcher) { + AtomicBoolean watchSet = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + v = new CollectionWatch(); + watchSet.set(true); + } + v.stateWatchers.add(stateWatcher); + return v; + }); + if (watchSet.get()) { + new StateWatcher(collection).refreshAndWatch(); synchronized (getUpdateLock()) { constructState(); } } } + /** + * Block until a CollectionStatePredicate returns true, or the wait times out + * + * Note that the predicate may be called again even after it has returned true, so + * implementors should avoid changing state within the predicate call itself. --- End diff -- I'm not talking about concurrent calls here, rather it's the race between adding the watcher, checking if the predicate matches the current state, and state changes arriving. There are two scenarios here (T1 = client thread, Z1 = zkCallback thread): T1 -> watcher added Z1 -> state changes, watcher called T1 -> watcher called for initial check Or, if we do things the other way round T1 -> watcher called for initial check and doesn't trigger Z1 -> state changes T1 -> watcher added, but isn't updated Absent adding locks all over the place, which I'm really reluctant to do here as it would be far too easy to end up in a deadlock, I think we have to accept that a predicate may get called twice. I don't think it matters too much though? This is generally just going to be used for waiting.

44. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61622572 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/CollectionStateWatcher.java --- @@ -0,0 +1,42 @@ +package org.apache.solr.common.cloud; + +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. +*/ + +import java.util.Set; + +/** + * Callback registered with {@link ZkStateReader#registerCollectionStateWatcher(String, CollectionStateWatcher)} + * and called whenever the collection state changes. + */ --- End diff -- Not sure! ¯_(ツ)_/¯
45. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61643539 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -1066,32 +1079,201 @@ public static String getCollectionPath(String coll) { return COLLECTIONS_ZKNODE+"/"+coll + "/state.json"; } - public void addCollectionWatch(String coll) { - if (interestingCollections.add(coll)) { - LOG.info("addZkWatch [{}]", coll); - new StateWatcher(coll).refreshAndWatch(false); + /** + * Notify this reader that a local Core is a member of a collection, and so that collection + * state should be watched. + * + * Not a public API. This method should only be called from ZkController. + * + * The number of cores per-collection is tracked, and adding multiple cores from the same + * collection does not increase the number of watches. + * + * @param collection the collection that the core is a member of + * + * @see ZkStateReader#unregisterCore(String) + */ + public void registerCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + reconstructState.set(true); + v = new CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new StateWatcher(collection).refreshAndWatch(); + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Notify this reader that a local core that is a member of a collection has been closed. + * + * Not a public API. This method should only be called from ZkController. + * + * If no cores are registered for a collection, and there are no {@link CollectionStateWatcher}s + * for that collection either, the collection watch will be removed. + * + * @param collection the collection that the core

belongs to + */ + public void unregisterCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) + return null; + if (v.coreRefCount > 0) + v.coreRefCount--; + if (v.canBeRemoved()) { + watchedCollectionStates.remove(collection); + lazyCollectionStates.put(collection, new LazyCollectionRef(collection)); + reconstructState.set(true); + return null; + } + return v; + }); + if (reconstructState.get()) { + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Register a CollectionStateWatcher to be called when the state of a collection changes + * + * + * A given CollectionStateWatcher will be only called once. If you want to have a persistent watcher, + * it should register itself again in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * method. + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher stateWatcher) { + AtomicBoolean watchSet = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + v = new CollectionWatch(); + watchSet.set(true); + } + v.stateWatchers.add(stateWatcher); + return v; + }); + if (watchSet.get()) { + new StateWatcher(collection).refreshAndWatch(); synchronized (getUpdateLock()) { constructState(); } } } + /** + * Block until a CollectionStatePredicate returns true, or the wait times out + * + * + * Note that the predicate may be called again even after it has returned true, so + * implementors should avoid changing state within the predicate call itself. --- End diff -- It seems like it would be nice to shield callers from doing any kind of similar mutexing. If you don't want to bother right now, I can come back and see if I can do something not yucky looking here.

46. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61643749 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/CollectionStatePredicate.java --- @@ -30,8 +30,9 @@ /** * Check the collection state matches a required state * - * The collectionState parameter may be null if the collection does not exist - * or has been deleted + * @param liveNodes the current set of live nodes + * @param collectionState the latest collection state, or null if the collection + * does not exist --- End diff -- I think this needs to be below the "Note" lines to get formatted right.
47. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61643877 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -256,9 +257,10 @@ public void updateClusterState() throws KeeperException, InterruptedException { refreshLegacyClusterState(null); // Need a copy so we don't delete from what we're iterating over. Collection<String> safeCopy = new ArrayList<>(watchedCollectionStates.keySet()); + Set<String> liveNodes = new HashSet<>(this.liveNodes); --- End diff -- You don't actually need a copy here, since `liveNodes` is an immutable set.
48. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61644721 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -491,19 +493,28 @@ private void refreshLegacyClusterState(Watcher watcher) final Stat stat = new Stat(); final byte[] data = zkClient.getData(CLUSTER_STATE, watcher, stat, true); final ClusterState loadedData = ClusterState.load(stat.getVersion(), data, emptySet(), CLUSTER_STATE); + final Set<String> liveNodes = new HashSet<>(this.liveNodes); synchronized (getUpdateLock()) { if (this.legacyClusterStateVersion >= stat.getVersion()) { // Nothing to do, someone else updated same or newer. return; } - this.legacyCollectionStates = loadedData.getCollectionStates(); - this.legacyClusterStateVersion = stat.getVersion(); - for (Map.Entry<String, ClusterState.CollectionRef> entry : this.legacyCollectionStates.entrySet()) { - if (entry.getValue().isLazilyLoaded() == false) { - // a watched collection - trigger notifications - notifyStateWatchers(entry.getKey(), entry.getValue().get()); + LOG.info("Updating legacy cluster state - {} entries in legacyCollectionStates", legacyCollectionStates.size()); + for (Map.Entry<String, CollectionWatch> watchEntry : this.collectionWatches.entrySet()) { + String coll = watchEntry.getKey(); + CollectionWatch collWatch = watchEntry.getValue(); + ClusterState.CollectionRef ref = this.legacyCollectionStates.get(coll); + if (ref == null) + continue; + // watched collection, so this will always be local + DocCollection newState = ref.get(); + if (!collWatch.stateWatchers.isEmpty() && + !Objects.equals(loadedData.getCollectionStates().get(coll).get(), newState)) { + notifyStateWatchers(liveNodes, coll, newState); --- End diff -- I just realized you don't want to call user code while holding the update lock. I think you're going to need to move this out of the synchronized block. In fact.... this is really nasty now that I think about it. In general, you're going to want to defer calling any user code until the current constuctState() operation finishes. Otherwise, the user code is potentially going to see a stale copy of the state that you haven't finished updating yet. I think we're going to have to build a queue of outstanding state watchers to notify and always call them later, probably in an

executor. I know that sounds like a bit of work, but I'm not sure I can see how it would be safe otherwise. @markrmiller any thoughts?

49. Pardon the distraction to the fine work going on here but I'd like to possibly emulate this code review process on other issue(s). Is it necessary to create a branch on some other/personal repo and then issue a pull request (as was done here I see) or is it possible for someone to review commits to a branch on our repo/mirror? I'm thinking SOLR-5750 -- https://github.com/apache/lucene-solr/commits/solr-5750 (feel free to make a comment to test).
50. Github user romseygeek commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r61777874 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -1066,32 +1079,201 @@ public static String getCollectionPath(String coll) { return COLLECTIONS_ZKNODE+"/"+coll + "/state.json"; } - public void addCollectionWatch(String coll) { - if (interestingCollections.add(coll)) { - LOG.info("addZkWatch [{}]", coll); - new StateWatcher(coll).refreshAndWatch(false); + /** + * Notify this reader that a local Core is a member of a collection, and so that collection + * state should be watched. + * + * Not a public API. This method should only be called from ZkController. + * + * The number of cores per-collection is tracked, and adding multiple cores from the same + * collection does not increase the number of watches. + * + * @param collection the collection that the core is a member of + * + * @see ZkStateReader#unregisterCore(String) + */ + public void registerCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + reconstructState.set(true); + v = new CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new StateWatcher(collection).refreshAndWatch(); + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Notify this reader that a local core that is a member of a collection has been closed. + * + * Not a public API. This method should only be called from ZkController. + * + * If no cores are registered for a collection, and there are no {@link CollectionStateWatcher}s + * for that collection either, the collection watch will be removed. + * + * @param collection the collection that the core belongs to + */ + public void unregisterCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) + return null; + if (v.coreRefCount > 0) + v.coreRefCount--; + if (v.canBeRemoved()) { + watchedCollectionStates.remove(collection); + lazyCollectionStates.put(collection, new LazyCollectionRef(collection)); + reconstructState.set(true); + return null; + } + return v; + }); + if (reconstructState.get()) { + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Register a CollectionStateWatcher to be called when the state of a collection changes + * + * A given CollectionStateWatcher will be only called once. If you want to have a persistent watcher, + * it should register itself again in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * method. + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher stateWatcher) { + AtomicBoolean watchSet = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + v = new CollectionWatch(); + watchSet.set(true); + } + v.stateWatchers.add(stateWatcher); + return v; + }); + if (watchSet.get()) { + new StateWatcher(collection).refreshAndWatch(); synchronized (getUpdateLock()) { constructState(); } } } + /** + * Block until a CollectionStatePredicate returns true, or the wait times out + * + * Note that the predicate may be called again even after it has returned true, so + * implementors should avoid changing state within the predicate call itself. --- End diff -- Yeah, I think this can be done in a follow-up issue, if need be?
51. Github user romseygeek commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-216315281 OK, latest push moves all notifications out of synchronized blocks.
52. Github user dragonsinth commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-216333963 @romseygeek nice job on the changes so far, and sorry to have so much feedback and so many asks. This is a pretty complicated change so I feel like it merits the attention to detail. I feel like we're at a fork in the road with this patch at the moment though, and we need to get more people involved to proceed. Let me explain. Even having fixed the "calling watchers while holding locks issue", the one thing that makes me most nervous about the current state is that we're still potentially executing user-provided predicates on threads that belong to a variety of other people-- e.g. the caller of forceUpdateCollection() or even the Zk event callback thread. We could make a tactical fix to the implementation of waitForState() by turning that method into a loop and running the predicate on the actual thread that called waitForState(), such that the onStateChanged() handler doesn't dip into client code. But honestly, I feel like having privatized CollectionStateWatcher and the ability to register / unregister is a missed opportunity. I can think of uses for the feature, like in some cases Overseer

operations could watch a collection for the duration of an operation to prevent having to re-query ZK. To make that solid, we'd need to either introduce an Executor in ZkStateReader for publishing events, or else require the watch registration to provide an executor, the way Guava's ListenableFuture does. Thoughts? I'd also like to hear from others.

53. Github user romseygeek commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-216347644 Feedback is good :-) I'll pull CSW back out and make it public again. I think keeping it separate from the Predicate is still a useful distinction though. I'll try adding in an executor as well.
54. Github user dragonsinth commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-216351404 BTW, here's an implementation of waitForState() that does the work on the calling thread. This passes your tests: ``` public void waitForState(final String collection, long wait, TimeUnit unit, CollectionStatePredicate predicate) throws InterruptedException, TimeoutException { long stop = System.nanoTime() + unit.toNanos(wait); if (predicate.matches(this.liveNodes, clusterState.getCollectionOrNull(collection))) { return; } LinkedBlockingQueue<Pair<Set<String>, DocCollection>> queue = new LinkedBlockingQueue<>(); CollectionStateWatcher watcher = new CollectionStateWatcher() { @Override public void onStateChanged(Set<String> liveNodes, DocCollection collectionState) { queue.add(new Pair<> (liveNodes, collectionState)); registerCollectionStateWatcher(collection, this); } }; registerCollectionStateWatcher(collection, watcher); try { while (true) { Pair<Set<String>, DocCollection> pair = queue.poll(stop - System.nanoTime(), TimeUnit.NANOSECONDS); if (pair == null) { throw new TimeoutException(); } if (predicate.matches(pair.getKey(), pair.getValue())) { return; } } } finally { removeCollectionStateWatcher(collection, watcher); } } ``` One thing I noticed in writing this is that it's uncertain whether you'll miss any states or not. I kind of like the idea that you could have your watcher return true or false to decide whether to keep watching, as it would ensure we could get all updates without missing any.
55. Github user romseygeek commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-216922168 > One thing I noticed in writing this is that it's uncertain whether you'll miss any states or not Unless I'm misunderstanding you, this is just how ZK works, though. A watcher firing just means that there has been at least one change to the watched node in the space of the last tick. So we wouldn't be able to guarantee that the CollectionStateWatcher is notified of every change. If we're making CSW public again, I don't think the queueing implementation you have there will work? An executor seems to be the most straightforward way of doing things here. Although, thinking more about that, we already have a separate executor for watchers, don't we? So this may just be overthinking things.
56. Github user dragonsinth commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-216931783 Correct, the queuing implementation where the waiting thread loops only helps waitForState(). Maybe we should just go with that for now and consider making CSW public as a follow up? If we do make it public, I think we'd still want a separate executor, you don't want to end up blocking ZKSR's internal operations.
57. Github user romseygeek commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-216941787 Last push exposes CollectionStateWatcher directly again, and moves notification calls into an Executor.
58. Github user markrmiller commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-217291207 > Although, thinking more about that, we already have a separate executor for watchers, don't we? Yes, every watch firing event should run from a dedicated executor rather than using ZK's event thread. I have not dug in enough here to know it covers what you guys are talking about, but holding up a Watcher thread should no longer interfere with ZK clients internal event thread.
59. Github user romseygeek commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-217978029 On further reflection, I've pulled the separate executor back out again. I think the SolrZkClient's separate executor will work well enough, and for the most part client code is going to be run inside a CloudSolrClient talking to the cluster, not a Solr node's internal state reader. I've also removed the CSC.registerCollectionWatcher() method, as that's really only for internal use, leaving the waitForState() method as a convenience. I'd like to commit this in the next couple of days. I think it will make a big difference to a bunch of test improvements I want to make separately.
60. Pretty sure you can create pull requests from branches within the same repository, so there's no need to have your own clone if you don't want one.
61. Github user dragonsinth commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-217984743 I did like the idea of a dedicated executor for collection events,

just to ensure clean separation. But I'll take a look in its current form.

62. Github user romseygeek commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-218122269 New plan! We now have a separate notification thread, and change notifications are placed into a LinkedBlockingQueue that the thread waits on. A caveat: the queue is currently unbounded, which is a Bad Thing. I'm not sure of the best way forward here - make it bounded, and just drop notifications if the queue is full? Make a note on the collection predicate javadocs that all predicates run in a single thread, and users need to be careful not to run slow code in them?
63. Github user dragonsinth commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-218226081 Hmm, isn't an executor a fancier way of doing a Queue + Thread(s)? :)
64. Github user romseygeek commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-218251017 That's... a good point, actually.
65. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62769742 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -485,6 +506,20 @@ private void refreshLegacyClusterState(Watcher watcher) // Nothing to do, someone else updated same or newer. return; } + Set<String> liveNodes = this.liveNodes; // volatile read + for (Map.Entry<String, CollectionWatch> watchEntry : this.collectionWatches.entrySet()) { + String coll = watchEntry.getKey(); + CollectionWatch collWatch = watchEntry.getValue(); + ClusterState.CollectionRef ref = this.legacyCollectionStates.get(coll); + if (ref == null) + continue; + // watched collection, so this will always be local --- End diff -- nit `legacy collections are always in-memory`
66. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62770082 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -485,6 +506,20 @@ private void refreshLegacyClusterState(Watcher watcher) // Nothing to do, someone else updated same or newer. return; } + Set<String> liveNodes = this.liveNodes; // volatile read + for (Map.Entry<String, CollectionWatch> watchEntry : this.collectionWatches.entrySet()) { + String coll = watchEntry.getKey(); + CollectionWatch collWatch = watchEntry.getValue(); + ClusterState.CollectionRef ref = this.legacyCollectionStates.get(coll); + if (ref == null) + continue; --- End diff -- Q: what happens if you try to set a watcher on a collection that doesn't exist yet? Mostly curious.
67. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62770248 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -635,6 +669,8 @@ public Object getUpdateLock() { public void close() { this.closed = true; + notifications.shutdownNow(); // interrupt --- End diff -- @markrmiller and I went on a hunt a while ago to try to remove most thread interruptions from Solr due to certain Lucene NIO operations getting permanently wedged due to interrupts. Is this necessary?
68. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62770279 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -635,6 +669,8 @@ public Object getUpdateLock() { public void close() { this.closed = true; + notifications.shutdownNow(); // interrupt + ExecutorUtil.shutdownAndAwaitTermination(notifications); --- End diff -- I think I would just shutdown and not wait.
69. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62770569 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -1069,32 +1100,190 @@ public static String getCollectionPath(String coll) { return COLLECTIONS_ZKNODE+"/"+coll + "/state.json"; } - public void addCollectionWatch(String coll) { - if (interestingCollections.add(coll)) { - LOG.info("addZkWatch [{}]", coll); - new StateWatcher(coll).refreshAndWatch(false); + /** + * Notify this reader that a local Core is a member of a collection, and so that collection + * state should be watched. + * + * Not a public API. This method should only be called from ZkController. + * + * The number of cores per-collection is tracked, and adding multiple cores from the same + * collection does not increase the number of watches. + * + * @param collection the collection that the core is a member of + * @see ZkStateReader#unregisterCore(String) + */ + public void registerCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + reconstructState.set(true); + v = new CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new StateWatcher(collection).refreshAndWatch(); + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Notify this reader that a local core that is a member of a collection has been closed. +

* + * Not a public API. This method should only be called from ZkController. + * + * If no cores are registered for a collection, and there are no {@link CollectionStateWatcher}s + * for that collection either, the collection watch will be removed. + * + * @param collection the collection that the core belongs to + */ + public void unregisterCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) + return null; + if (v.coreRefCount > 0) + v.coreRefCount--; + if (v.canBeRemoved()) { + watchedCollectionStates.remove(collection); + lazyCollectionStates.put(collection, new LazyCollectionRef(collection)); + reconstructState.set(true); + return null; + } + return v; + }); + if (reconstructState.get()) { + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Register a CollectionStateWatcher to be called when the state of a collection changes + * + * + * A given CollectionStateWatcher will be only called once. If you want to have a persistent watcher, + * it should register itself again in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * method. + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher stateWatcher) { + AtomicBoolean watchSet = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + v = new CollectionWatch(); + watchSet.set(true); + } + v.stateWatchers.add(stateWatcher); + return v; + }); + if (watchSet.get()) { + new StateWatcher(collection).refreshAndWatch(); synchronized (getUpdateLock()) { constructState(); } } } + /** + * Block until a CollectionStatePredicate returns true, or the wait times out + * + * + * Note that the predicate may be called again even after it has returned true, so + * implementors should avoid changing state within the predicate call itself. + * + * @param collection the collection to watch + * @param wait how long to wait + * @param unit the units of the wait parameter + * @param predicate the predicate to call on state changes + * @throws InterruptedException on interrupt + * @throws TimeoutException on timeout + */ + public void waitForState(final String collection, long wait, TimeUnit unit, CollectionStatePredicate predicate) + throws InterruptedException, TimeoutException { + + final CountDownLatch latch = new CountDownLatch(1); + + CollectionStateWatcher watcher = new CollectionStateWatcher() { + @Override + public void onStateChanged(Set<String> liveNodes, DocCollection collectionState) { + if (predicate.matches(liveNodes, collectionState)) { + latch.countDown(); + } else { + registerCollectionStateWatcher(collection, this); + } + } + }; + registerCollectionStateWatcher(collection, watcher); + + try { + // check the current state + DocCollection dc = clusterState.getCollectionOrNull(collection); + if (predicate.matches(liveNodes, dc)) + return; + + // wait for the watcher predicate to return true, or time out + if (!latch.await(wait, unit)) + throw new TimeoutException(); + + } + finally { --- End diff -- nit: you have a few formatting issues here and elsewhere

70. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62770640 --- Diff: solr/solrj/src/java/org/apache/solr/common/util/ExecutorUtil.java --- @@ -154,6 +147,20 @@ public static ExecutorService newMDCAwareSingleThreadExecutor(ThreadFactory thre } /** + * Create a single thread executor using a named thread factory + */ + public static ExecutorService newMDCAwareSingleThreadExecutor(String name) { + return newMDCAwareSingleThreadExecutor(new SolrjNamedThreadFactory(name)); + } --- End diff -- not used
71. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62771056 --- Diff: solr/solrj/src/test/org/apache/solr/common/cloud/TestCollectionStateWatchers.java --- @@ -0,0 +1,235 @@ +package org.apache.solr.common.cloud; + +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +import java.util.HashMap; +import java.util.Set; +import java.util.concurrent.CountDownLatch; +import java.util.concurrent.ExecutorService; +import java.util.concurrent.Future; +import java.util.concurrent.TimeUnit; +import java.util.concurrent.TimeoutException; +import java.util.concurrent.atomic.AtomicInteger; + +import org.apache.solr.client.solrj.embedded.JettySolrRunner; +import org.apache.solr.client.solrj.impl.CloudSolrClient; +import

org.apache.solr.client.solrj.request.CollectionAdminRequest; +import org.apache.solr.cloud.SolrCloudTestCase; +import org.apache.solr.common.util.ExecutorUtil; +import org.apache.solr.common.util.SolrjNamedThreadFactory; +import org.junit.AfterClass; +import org.junit.Before; +import org.junit.BeforeClass; +import org.junit.Test; + +import static org.hamcrest.core.Is.is; + +public class TestCollectionStateWatchers extends SolrCloudTestCase { + + private static final int CLUSTER_SIZE = 4; + + private static final ExecutorService executor = ExecutorUtil.newMDCAwareCachedThreadPool(+ new SolrjNamedThreadFactory("backgroundWatchers")); + + private static final int MAX_WAIT_TIMEOUT = 30; + + @BeforeClass + public static void startCluster() throws Exception { + configureCluster(CLUSTER_SIZE) + .addConfig("config", getFile("solrj/solr/collection1/conf").toPath()) + .configure(); + } + + @AfterClass + public static void shutdownBackgroundExecutors() { + executor.shutdown(); + } + + @Before + public void prepareCluster() throws Exception { + int missingServers = CLUSTER_SIZE - cluster.getJettySolrRunners().size(); + for (int i = 0; i < missingServers; i++) { + cluster.startJettySolrRunner(); + } + cluster.waitForAllNodes(30); + } + + private static Future<Boolean> waitInBackground(String collection, long timeout, TimeUnit unit, + CollectionStatePredicate predicate) { + return executor.submit(() -> { + try { + cluster.getSolrClient().waitForState(collection, timeout, unit, predicate); + } catch (InterruptedException | TimeoutException e) { + return Boolean.FALSE; + } + return Boolean.TRUE; + }); + } + + + @Test + public void testSimpleCollectionWatch() throws Exception { + + CloudSolrClient client = cluster.getSolrClient(); + cluster.createCollection("testcollection", CLUSTER_SIZE, 1, "config", new HashMap<>()); + + client.waitForState("testcollection", MAX_WAIT_TIMEOUT, TimeUnit.SECONDS, DocCollection::isFullyActive); + + // shutdown a node and check that we get notified about the change + final AtomicInteger nodeCount = new AtomicInteger(0); + final CountDownLatch latch = new CountDownLatch(1); + client.registerCollectionStateWatcher("testcollection", (liveNodes, collectionState) -> { + // we can't just count liveNodes here, because that's updated by a separate watcher, + // and it may be the case that we're triggered by a node setting itself to DOWN before + // the liveNodes watcher is called + for (Slice slice : collectionState) { + for (Replica replica : slice) { + if (replica.isActive(liveNodes)) + nodeCount.incrementAndGet(); + } + } + latch.countDown(); + }); + + cluster.stopJettySolrRunner(random().nextInt(cluster.getJettySolrRunners().size())); + assertTrue("CollectionStateWatcher was never notified of cluster change", latch.await(MAX_WAIT_TIMEOUT, TimeUnit.SECONDS)); + + assertThat(nodeCount.intValue(), is(3)); + + } + + @Test + public void testWaitForStateChecksCurrentState() throws Exception { + + CloudSolrClient client = cluster.getSolrClient(); + cluster.createCollection("waitforstate", 1, 1, "config", new HashMap<>()); + + client.waitForState("waitforstate", MAX_WAIT_TIMEOUT, TimeUnit.SECONDS, DocCollection::isFullyActive); + + // several goes, to check that we're not getting delayed state changes + for (int i = 0; i < 10; i++) { + try { + client.waitForState("waitforstate", 1, TimeUnit.SECONDS, DocCollection::isFullyActive); + } catch (TimeoutException e) { + fail("waitForState should return immediately if the predicate is already satisfied"); + } + } + + } + + @Test + public void testCanWatchForNonexistantCollection() throws Exception { { --- End diff -- technical can _wait_ for non-existent collection. Would like to know what happens if you try to watch for a non-existent one, does it get fired immediately, or only when the collection is created?

72. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62771165 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -1069,32 +1100,190 @@ public static String getCollectionPath(String coll) { return COLLECTIONS_ZKNODE+"/"+coll + "/state.json"; } - public void addCollectionWatch(String coll) { - if (interestingCollections.add(coll)) { - LOG.info("addZkWatch [{}]", coll); - new StateWatcher(coll).refreshAndWatch(false); + /** + * Notify this reader that a local Core is a member of a collection, and so that collection + * state should be watched. + * + * Not a public API. This method should only be called from ZkController. + * + * The number of cores per-collection is tracked, and adding multiple cores from the same + * collection does not increase the number of watches. + * + * @param collection the collection that the core is a member of + * + * @see ZkStateReader#unregisterCore(String) + */ + public void registerCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + reconstructState.set(true); + v = new CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new StateWatcher(collection).refreshAndWatch(); + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Notify this reader that a local core that is a member of a collection has been closed. + * + * Not a public API. This method should only be called from ZkController. + * + * If no cores are

registered for a collection, and there are no {@link CollectionStateWatcher}s + * for that collection either, the collection watch will be removed. + * + * @param collection the collection that the core belongs to + */ + public void unregisterCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) + return null; + if (v.coreRefCount > 0) + v.coreRefCount--; + if (v.canBeRemoved()) { + watchedCollectionStates.remove(collection); + lazyCollectionStates.put(collection, new LazyCollectionRef(collection)); + reconstructState.set(true); + return null; + } + return v; + }); + if (reconstructState.get()) { + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Register a CollectionStateWatcher to be called when the state of a collection changes + * + * + * A given CollectionStateWatcher will be only called once. If you want to have a persistent watcher, + * it should register itself again in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * method. + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher stateWatcher) { + AtomicBoolean watchSet = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + v = new CollectionWatch(); + watchSet.set(true); + } + v.stateWatchers.add(stateWatcher); + return v; + }); + if (watchSet.get()) { + new StateWatcher(collection).refreshAndWatch(); --- End diff -- I feel like this needs to check whether or not the collection exists / is a legacy collection. If it's a legacy collection, you don't want to try to create a StateWatcher.

73. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62771180 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -1069,32 +1100,190 @@ public static String getCollectionPath(String coll) { return COLLECTIONS_ZKNODE+"/"+coll + "/state.json"; } - public void addCollectionWatch(String coll) { - if (interestingCollections.add(coll)) { - LOG.info("addZkWatch [{}]", coll); - new StateWatcher(coll).refreshAndWatch(false); + /** + * Notify this reader that a local Core is a member of a collection, and so that collection + * state should be watched. + * + * Not a public API. This method should only be called from ZkController. + * + * The number of cores per-collection is tracked, and adding multiple cores from the same + * collection does not increase the number of watches. + * + * @param collection the collection that the core is a member of + * + * @see ZkStateReader#unregisterCore(String) + */ + public void registerCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + reconstructState.set(true); + v = new CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new StateWatcher(collection).refreshAndWatch(); --- End diff -- (same) I feel like this needs to check whether or not the collection exists / is a legacy collection. If it's a legacy collection, you don't want to try to create a StateWatcher.
74. Github user dragonsinth commented on the pull request: https://github.com/apache/lucene-solr/pull/32#issuecomment-218321437 Almost LGTM. There's a few nits, but the only real issue is potentially setting up a StateWatcher on legacy. Nice work, I think we're almost done!!
75. Github user romseygeek commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62810535 --- Diff: solr/solrj/src/test/org/apache/solr/common/cloud/TestCollectionStateWatchers.java --- @@ -0,0 +1,235 @@ +package org.apache.solr.common.cloud; + +/* + * Licensed to the Apache Software Foundation (ASF) under one or more + * contributor license agreements. See the NOTICE file distributed with + * this work for additional information regarding copyright ownership. + * The ASF licenses this file to You under the Apache License, Version 2.0 + * (the "License"); you may not use this file except in compliance with + * the License. You may obtain a copy of the License at + * + * http://www.apache.org/licenses/LICENSE-2.0 + * + * Unless required by applicable law or agreed to in writing, software + * distributed under the License is distributed on an "AS IS" BASIS, + * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. + * See the License for the specific language governing permissions and + * limitations under the License. + */ + +import java.util.HashMap; +import java.util.Set; +import java.util.concurrent.CountDownLatch; +import java.util.concurrent.ExecutorService; +import java.util.concurrent.Future; +import java.util.concurrent.TimeUnit; +import java.util.concurrent.TimeoutException; +import java.util.concurrent.atomic.AtomicInteger; + +import org.apache.solr.client.solrj.embedded.JettySolrRunner; +import org.apache.solr.client.solrj.impl.CloudSolrClient; +import org.apache.solr.client.solrj.request.CollectionAdminRequest; +import org.apache.solr.cloud.SolrCloudTestCase; +import org.apache.solr.common.util.ExecutorUtil; +import org.apache.solr.common.util.SolrjNamedThreadFactory; +import org.junit.AfterClass; +import

org.junit.Before; +import org.junit.BeforeClass; +import org.junit.Test; + +import static org.hamcrest.core.Is.is; + +public class TestCollectionStateWatchers extends SolrCloudTestCase { + + private static final int CLUSTER_SIZE = 4; + + private static final ExecutorService executor = ExecutorUtil.newMDCAwareCachedThreadPool(+ new SolrjNamedThreadFactory("backgroundWatchers")); + + private static final int MAX_WAIT_TIMEOUT = 30; + + @BeforeClass + public static void startCluster() throws Exception { + configureCluster(CLUSTER_SIZE) + .addConfig("config", getFile("solrj/solr/collection1/conf").toPath()) + .configure(); + } + + @AfterClass + public static void shutdownBackgroundExecutors() { + executor.shutdown(); + } + + @Before + public void prepareCluster() throws Exception { + int missingServers = CLUSTER_SIZE - cluster.getJettySolrRunners().size(); + for (int i = 0; i < missingServers; i++) { + cluster.startJettySolrRunner(); + } + cluster.waitForAllNodes(30); + } + + private static Future<Boolean> waitInBackground(String collection, long timeout, TimeUnit unit, + CollectionStatePredicate predicate) { + return executor.submit(() -> { + try { + cluster.getSolrClient().waitForState(collection, timeout, unit, predicate); + } catch (InterruptedException | TimeoutException e) { + return Boolean.FALSE; + } + return Boolean.TRUE; + }); + } + + + @Test + public void testSimpleCollectionWatch() throws Exception { + + CloudSolrClient client = cluster.getSolrClient(); + cluster.createCollection("testcollection", CLUSTER_SIZE, 1, "config", new HashMap<>()); + + client.waitForState("testcollection", MAX_WAIT_TIMEOUT, TimeUnit.SECONDS, DocCollection::isFullyActive); + + // shutdown a node and check that we get notified about the change + final AtomicInteger nodeCount = new AtomicInteger(0); + final CountDownLatch latch = new CountDownLatch(1); + client.registerCollectionStateWatcher("testcollection", (liveNodes, collectionState) -> { + // we can't just count liveNodes here, because that's updated by a separate watcher, + // and it may be the case that we're triggered by a node setting itself to DOWN before + // the liveNodes watcher is called + for (Slice slice : collectionState) { + for (Replica replica : slice) { + if (replica.isActive(liveNodes)) + nodeCount.incrementAndGet(); + } + } + latch.countDown(); + }); + + cluster.stopJettySolrRunner(random().nextInt(cluster.getJettySolrRunners().size())); + assertTrue("CollectionStateWatcher was never notified of cluster change", latch.await(MAX_WAIT_TIMEOUT, TimeUnit.SECONDS)); + + assertThat(nodeCount.intValue(), is(3)); + + } + + @Test + public void testWaitForStateChecksCurrentState() throws Exception { + + CloudSolrClient client = cluster.getSolrClient(); + cluster.createCollection("waitforstate", 1, 1, "config", new HashMap<>()); + + client.waitForState("waitforstate", MAX_WAIT_TIMEOUT, TimeUnit.SECONDS, DocCollection::isFullyActive); + + // several goes, to check that we're not getting delayed state changes + for (int i = 0; i < 10; i++) { + try { + client.waitForState("waitforstate", 1, TimeUnit.SECONDS, DocCollection::isFullyActive); + } catch (TimeoutException e) { + fail("waitForState should return immediately if the predicate is already satisfied"); + } + } + + } + + @Test + public void testCanWatchForNonexistantCollection() throws Exception { --- End diff -- If there's already a watch set, then it won't be fired. If there's no watch set, then it will be fired by StateWatcher#refreshAndWatch().

76. Github user romseygeek commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62810671 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -635,6 +669,8 @@ public Object getUpdateLock() { public void close() { this.closed = true; + notifications.shutdownNow(); // interrupt --- End diff -- Ah, no, that's left over from the queue impl, will remove it.
77. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62900820 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -1069,32 +1100,190 @@ public static String getCollectionPath(String coll) { return COLLECTIONS_ZKNODE+"/"+coll + "/state.json"; } - public void addCollectionWatch(String coll) { - if (interestingCollections.add(coll)) { - LOG.info("addZkWatch [{}]", coll); - new StateWatcher(coll).refreshAndWatch(false); + /** + * Notify this reader that a local Core is a member of a collection, and so that collection + * state should be watched. + * + * Not a public API. This method should only be called from ZkController. + * + * The number of cores per-collection is tracked, and adding multiple cores from the same + * collection does not increase the number of watches. + * + * @param collection the collection that the core is a member of + * + * @see ZkStateReader#unregisterCore(String) + */ + public void registerCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + reconstructState.set(true); + v = new CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new

StateWatcher(collection).refreshAndWatch(); --- End diff -- Ignore this, I'm dumb. You want a state watcher either way (the old code did this).

78. Github user dragonsinth commented on a diff in the pull request: https://github.com/apache/lucene-solr/pull/32#discussion_r62900842 --- Diff: solr/solrj/src/java/org/apache/solr/common/cloud/ZkStateReader.java --- @@ -1069,32 +1100,190 @@ public static String getCollectionPath(String coll) { return COLLECTIONS_ZKNODE+"/"+coll + "/state.json"; } - public void addCollectionWatch(String coll) { - if (interestingCollections.add(coll)) { - LOG.info("addZkWatch [{}]", coll); - new StateWatcher(coll).refreshAndWatch(false); + /** + * Notify this reader that a local Core is a member of a collection, and so that collection + * state should be watched. + * + * Not a public API. This method should only be called from ZkController. + * + * The number of cores per-collection is tracked, and adding multiple cores from the same + * collection does not increase the number of watches. + * + * @param collection the collection that the core is a member of + * + * @see ZkStateReader#unregisterCore(String) + */ + public void registerCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + reconstructState.set(true); + v = new CollectionWatch(); + } + v.coreRefCount++; + return v; + }); + if (reconstructState.get()) { + new StateWatcher(collection).refreshAndWatch(); + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Notify this reader that a local core that is a member of a collection has been closed. + * + * Not a public API. This method should only be called from ZkController. + * + * If no cores are registered for a collection, and there are no {@link CollectionStateWatcher}s + * for that collection either, the collection watch will be removed. + * + * @param collection the collection that the core belongs to + */ + public void unregisterCore(String collection) { + AtomicBoolean reconstructState = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) + return null; + if (v.coreRefCount > 0) + v.coreRefCount--; + if (v.canBeRemoved()) { + watchedCollectionStates.remove(collection); + lazyCollectionStates.put(collection, new LazyCollectionRef(collection)); + reconstructState.set(true); + return null; + } + return v; + }); + if (reconstructState.get()) { + synchronized (getUpdateLock()) { + constructState(); + } + } + } + + /** + * Register a CollectionStateWatcher to be called when the state of a collection changes + * + * A given CollectionStateWatcher will be only called once. If you want to have a persistent watcher, + * it should register itself again in its {@link CollectionStateWatcher#onStateChanged(Set, DocCollection)} + * method. + */ + public void registerCollectionStateWatcher(String collection, CollectionStateWatcher stateWatcher) { + AtomicBoolean watchSet = new AtomicBoolean(false); + collectionWatches.compute(collection, (k, v) -> { + if (v == null) { + v = new CollectionWatch(); + watchSet.set(true); + } + v.stateWatchers.add(stateWatcher); + return v; + }); + if (watchSet.get()) { + new StateWatcher(collection).refreshAndWatch(); --- End diff -- ditto, ignore this
79. Final patch. I think this is ready!
80. I already LGTM'd the github PR, I don't think I need to look at the patch file?
81. Commit b6d742141250a8395c96d364714a31f4a3a63a96 in lucene-solr's branch refs/heads/master from [~romseygeek] [https://git-wip-us.apache.org/repos/asf?p=lucene-solr.git;h=b6d7421] SOLR-8323: Add CollectionStateWatcher API
82. Commit 06d2f6368df9b6d29d852f18bab38d96255d83c7 in lucene-solr's branch refs/heads/branch_6x from [~romseygeek] [https://git-wip-us.apache.org/repos/asf?p=lucene-solr.git;h=06d2f63] SOLR-8323: Add CollectionStateWatcher API
83. Thanks for all the reviewing Scott! Now on to SOLR-9056 :)
84. Commit 963c6522b6e10bfeaad340457d1e96351d0aecc1 in lucene-solr's branch refs/heads/master from [~romseygeek] [https://git-wip-us.apache.org/repos/asf?p=lucene-solr.git;h=963c652] SOLR-8323: DocCollection.isFullyActive needs to know how many replicas to expect
85. Commit 06ebd4fd7e9045b28a6a243c56f753b4f56c8561 in lucene-solr's branch refs/heads/branch_6x from [~romseygeek] [https://git-wip-us.apache.org/repos/asf?p=lucene-solr.git;h=06ebd4f] SOLR-8323: DocCollection.isFullyActive needs to know how many replicas to expect
86. Commit c0d23a741e9f2c787ab322e29c67108e5fd5c692 in lucene-solr's branch refs/heads/master from [~romseygeek] [https://git-wip-us.apache.org/repos/asf?p=lucene-solr.git;h=c0d23a7] SOLR-8323: Handle removal of legacy collections
87. Commit b5c369a773689955aa9bbd1b0bce3b7d1d96cb1b in lucene-solr's branch refs/heads/branch_6x from [~romseygeek] [https://git-wip-us.apache.org/repos/asf?p=lucene-solr.git;h=b5c369a] SOLR-8323: Handle removal of legacy collections
88. Still seeing test failures here, for example: http://jenkins.thetaphi.de/job/Lucene-Solr-6.x-Windows/181/consoleFull It looks as though the collection watcher is being fired even though the state hasn't actually changed. Will add some more debugging to try and work out why.

89. OK, I think I see what's happening. The test waits for a collection to be up, and then registers a watch to check for subsequent changes. Once a wait has returned, then the watch is removed, but the collection isn't actually removed from the 'interesting' list until the next state update. We have a race between state watchers being cleared after firing and the subsequent removal of a collection from the 'interesting' list, and the new watcher being added. On a fast machine, the new watcher is added before the old one is cleared, and so the state is preserved between the calls; this means that the new watcher isn't actually fired, because we check if the state has changed before running notifications. On a slower machine, the watcher is added afterwards, so there's no previous state to compare against, so the watcher is fired immediately - resulting in a test failure, because the test is expecting the state to have changed. We can fix the test by calling registerCore() first, which puts the collection permanently on the watched list, but I think there's a bigger question here about how useful statewatchers are, as opposed to state predicate checks. All the uses I've come up with so far have just been waitForState() calls. [~dragonsinth] what do you think?
90. **body:** [~romseygeek] sorry I'm just now having a moment to look at this. Is this still flakey on master? Do you have a good spot to drop in a Thread.sleep() to trigger this reliably? If not I'll play around.
    label: test
91. SOLR-9113 fixed the test failures.
92. Nice, I was just figuring that out. :D