Item 323
git_comments:

1. our state is up to date
2. * 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.
3. collection does not exist
4. * * Refresh the Cluster State for a given collection *
5. * * Submit an intra-process message which will be picked up and executed when {@link ClusterStateUpdater}'s * loop runs next time
6. Only the elements that were asked for... ...get added to the map
7. Now ask Overseer to fetch the latest state of collection from ZK
8. we have successfully found all replicas to be ACTIVE
9. In case of a PRS collection, create the collection structure directly instead of resubmitting to the overseer queue. TODO: Consider doing this for all collections, not just the PRS collections.
10. could be a big cluster
11. log.info("collection updated : {}", new String(data, StandardCharsets.UTF_8));
12. In case of a PRS collection, execute the ADDREPLICA directly instead of resubmitting to the overseer queue. TODO: Consider doing this for all collections, not just the PRS collections.
13. * Check and return if all replicas are ACTIVE
14. Now let's do an add replica

git_commits:

1. **summary:** SOLR-15138: Collection creation for PerReplicaStates does not scale to large collections as well as regular collections (#2318)
   message: SOLR-15138: Collection creation for PerReplicaStates does not scale to large collections as well as regular collections (#2318)

github_issues:

github_issues_comments:

github_pulls:

1. **title:** SOLR-15138: PerReplicaStates does not scale to large collections as well as state.json
   body: WIP

github_pulls_comments:

1. > Error is a timeout from `CollectionsHandler` having waited 45 seconds I take that back @noblepaul . I did the test wrong and just did it again and it passes. Timing for the collection creation (11x11=121 replicas on 3 nodes) is similar with or without PRS at about 45 seconds. I can do more testing later (more concurrent threads, more and smaller collections). Note I did put out a few numbers on PRS (not with the patch in this PR though), see [this comment](https://issues.apache.org/jira/browse/SOLR-15146?focusedCommentId=17281460&page=com.atlassian.jira.plugin.system.issuetabpanels%3Acomment-tabpanel#comment-17281460).
2. Thanks for your review, @murblanc. I learned a lot from your detailed review comment. I've added a commit to let the overseer thread refresh the view of the cluster state immediately after the collection is created. Please review. CC @noblepaul.
3. > Thanks for your review, @murblanc. I learned a lot from your detailed review comment. Thank you! > I've added a commit to let the overseer thread refresh the view of the cluster state immediately after the collection is created. Please review. CC @noblepaul. I have very serious doubts about the proposed

change. Forcing a cluster state updater refresh in that way is going to force it to re-read all relevant ZK data through `forciblyRefreshAllClusterStateSlow()` as it does during Overseer start up, stalling processing of the state update queue in the meantime. And this will happen each time a new PRS collection is created. And it is not even sufficient... If you look at `ClusterStateUpdater.run()`, that flag was really meant to be used once at the beginning. There are two processing loops, and the inner loop happens after the refresh and will not look at the flag, so some processing can take place before the flag is even considered. So it may well be that the PRS collection is created, the flag is set on the Overseer, the creation call returns, client sends another Collection API related to the collection but `ClusterStateUpdater` hasn't updated yet from ZK (hasn't even started to update). This type of race would in theory be possible even without the inner loop given the batching done by the cluster state updater (+ the waiting time to see if new messages arrive in order to batch them together). I really feel that a proper refactoring of what gets executed and where it gets executed is a better option (esp. for the master branch). Why not submit the `state.json` through the Overseer as happens for non PRS collections then create the PRS znodes in the Collection API call to not stall the cluster state updater for too long? If you really have to force an Overseer `ClusterStateUpdater` refresh for that collection anyway, introducing a new type of cluster state updater message (in `OverseerAction`) called `LOADCOLLECTION` for example that triggers the loading of that single collection from ZK into the state used by the cluster state updater seems to me a better option. I don't like it either, but I hate it less. Note the client will see correct behavior only if it then submits the collection creation call and wait for it to return before submitting another Collection API call for that collection, to make sure that the `LOADCOLLECTION` gets executed first (with current Solr client can submit collection creation immediately followed by another Collection API call for that collection and they will be executed in order).
4. > I have very serious doubts about the proposed change. > Forcing a cluster state updater refresh in that way is going to force it to re-read all relevant ZK data through forciblyRefreshAllClusterStateSlow() as it does during Overseer start up, stalling processing of the state update queue in the meantime. And this will happen each time a new PRS collection is created. Agree, this is incredibly inefficient. I'm reverting the commit for now.
5. > Once the operational bugs are fixed, in my opinion we should consider this code as stop gap, and eventually redo things in a cleaner and more efficient way at some later time (ideally before Solr 9 is out). Maintaining this code as it is is not going to be fun. I agree. I think we should phase out non-PRS mode altogether soon, and then all of this won't be a burden to maintain.
6. I tested this using the stress test suite, and performance for collection creations are at par with non-PRS collections and I saw no timeouts/failures etc. I think this change is good to go with.

github_pulls_reviews:

1. `AtomicBoolean` would likely make code easier to read.
2. This method caches the value computed the first time it's called, then returns that value on each subsequent call. When used as part of the predicate in `CollectionStateWatcher`, even if the watch fires multiple times wouldn't it always return the first computed state (which is likely `false` as all replicas are not active initially)?
3. Truncated comment
4. Shouldn't that test about collection not being PRS be made before the call to `waitForState` to just skip that call?
5. This object is immutable. Everytime the state is modified, a new instance is created
6. could have. But, it will return as soon as the cal is made.
7. I agree with Ilan here, let's skip the extra watcher and call to ZK.
8. Agree.
9. This call does nothing and can be removed. The command already has the instance of `ClusterState` returned by the call (it is passed to it as a `call` parameter by OCMH). The only way the clusterState is refreshed is when the watchers on the node get called and fetch the latest updates from ZK (that in this case were done by the Overseer, and we know the watchers fired and updated our local state because we've waited for it in the `while` loop ~10 lines above).
10. Calling this method here (or more specifically using its output in the `create()` call below) breaks the cluster state updater contract in Overseer. This call updates a `state.json` from a thread other than the cluster state updater thread in Overseer, the cluster state updater will not know about the new Json being written (the fact that this thread running this method is executing on the Overseer node is irrelevant, except passing stats, cluster state updater and Collection API execution are totally independent and could be running on different nodes). Overseer has two copies of the cluster state: the cluster state maintained in

`ZkStateReader` like any node has (this is the state passed to Collection API calls), and another copy of the state maintained in `ZkStateWriter` which initially is obtained from `ZkStateReader` but quickly diverges and then used as the "source of truth" by the cluster state updater that keeps writing back that state to Zookeeper for the whole life of the Overseer on the node. The cluster updater state has no watches is only updated by the cluster state updater when processing messages and then the updated value written to ZK. To achieve the effect of dealing with the Per Replica States directly from this thread rather than from the ClusterStateUpdater single thread, the state.json would have to be created by the Overseer (i.e. through a classic `ocmh.overseer.offerStateUpdate(Utils.toJSON(message));`) and once done the PRS could be created from here. This is complicated to do because of the way `ZkStateWriter` is implemented to write the `state.json` as well as handle the PRS operations in the same method (when it calls `cmd.ops.persist` in `writePendingUpdates()`) and would require quite some refactoring.
11. This call bypasses the cluster state updater to update the `state.json`. If it tried to do so through the Overseer cluster state updater, the update would fail as the collection is unknown there.
12. When we get here for a PRS collection, the `ZkStateReader` cluster state for the collection is fine, as are the ZK structures, but the Overseer cluster state updater does not know about the collection. Any new operation on the collection through the Collection API would fail in the cluster state updater.
13. please please please configure your IDE to not do this. it's been an issue on other PRs as well.
14. This section looks like almost a duplicate of the other `if(isPrs)` block earlier. No concrete reason, but this doesn't seem right.
15. We're doing this for each replica individually, which turns into a lot more writes. Previously the overseer would batch up the updates, I believe, and minimize the number of updates to state.json. Can we do a ZK multi-operation here instead of lots of small writes in a tight loop?
16. Can we set a watch (maybe on an arbitrary down replica?) instead of doing a loop+sleep?
17. yes
18. Can you please use another name than `run()` that is usually related to a `Runnable` which is not the case here.
19. No attributes of `Message` are used in the code. The queue could well be defined as `CopyOnWriteArrayList<Object>` and not a single line of code would have to change. I suggest we define the processing method in Message (`processMessage`? the one currently called `run`) and when getting a new message from the queue above in `Message m = unprocessedMessages.remove(0);` simply call `m.processMessage(clusterState, Overseer.this);`
20. Note that each call to `addReplica` here will read back all existing znodes under `state.json`. This is therefore running in `n^2` (`n*(n+1)/2`) with `n` the number of replicas. (`fetch` in `PerReplicaStates` called from `SliceMutator.addReplica` will always see a different `cversion` value since the previous replica was added on the previous iteration and will therefore `getChildren`)
21. move comment into the `else` bloc.
22. In case of PRS collection (and failure, so we're executing here), the collection delete called from two lines below (cannot attach comment to the actual line, thanks GitHub) will fail because the `ClusterStateUpdater` will not know about the collection, it wasn't refreshed. The failure will happen while waiting for the state after the call to `ocmh.overseer.offerStateUpdate(Utils.toJSON(m));` in `DeleteCollectionCmd`.
23. I can confirm that this is no longer an issue after the latest commits. I ran CreateCollectionCleanupTest with the following patch [0] and it passed consistently. While it ran, I checked the coverage report to verify that these lines were covered. [0] - https://paste.centos.org/view/09e3434d
24. ^ My above comment was based on the 8x change, and seems like this change got missed when porting them over to this PR (for master). I'll update this branch and bring it up to sync with 8x soon.
25. This is resolved now.

jira_issues:

jira_issues_comments: