Item 56
git_comments:

git_commits:

1. **summary:** KAFKA-9089; Reassignment should be resilient to unexpected errors (#7562)
   message: KAFKA-9089; Reassignment should be resilient to unexpected errors (#7562) The purpose of this patch is to make the reassignment algorithm simpler and more resilient to unexpected errors. Specifically, it has the following improvements: 1. Remove `ReassignedPartitionContext`. We no longer need to track the previous reassignment through the context and we now use the assignment state as the single source of truth for the target replicas in a reassignment. 2. Remove the intermediate assignment state when overriding a previous reassignment. Instead, an overriding reassignment directly updates the assignment state and shuts down any unneeded replicas. Reassignments are _always_ persisted in Zookeeper before being updated in the controller context. 3. To fix race conditions with concurrent submissions, reassignment completion for a partition always checks for a zk partition reassignment to be removed. This means the controller no longer needs to track the source of the reassignment. 4. Api reassignments explicitly remove reassignment state from zk prior to beginning the new reassignment. This fixes an inconsistency in precedence. Upon controller failover, zookeeper reassignments always take precedence over any active reassignment. So if we do not have the logic to remove the zk reassignment when an api reassignment is triggered, then we can revert to the older zk reassignment. Reviewers: Viktor Somogyi <viktorsomogyi@gmail.com>, Stanislav Kozlovski <stanislav_kozlovski@outlook.com>, Jun Rao <junrao@gmail.com>

github_issues:

github_issues_comments:

github_pulls:

1. **title:** KAFKA-9089; Reassignment should be resilient to unexpected errors
   body: The purpose of this patch is to make the reassignment algorithm simpler and more resilient to unexpected errors. Specifically, it has the following improvements: 1. Remove `ReassignedPartitionContext`. We no longer need to track the previous reassignment through the context and we now use the assignment state as the single source of truth for the target replicas in a reassignment. 2. Remove the intermediate assignment state when overriding a previous reassignment. Instead, an overriding reassignment directly updates the assignment state and shuts down any unneeded replicas. Reassignments are _always_ persisted in Zookeeper before being updated in the controller context. 3. To fix race conditions with concurrent submissions, reassignment completion for a partition always checks for a zk partition reassignment to be removed. This means the controller no longer needs to track the source of the reassignment. 4. Api reassignments explicitly remove reassignment state from zk prior to beginning the new reassignment. This fixes an inconsistency in precedence. Upon controller failover, zookeeper reassignments always take precedence over any active reassignment. So if we do not have the logic to remove the zk reassignment when an api reassignment is triggered, then we can revert to the older zk reassignment. ### Committer Checklist (excluded from commit message) - [ ] Verify design and implementation - [ ] Verify test coverage and CI build status - [ ] Verify documentation (including upgrade notes)

github_pulls_comments:

1. cc @stanislavkozlovski Can you take a close look at the updates? I think the main changes are the following: - I've removed the intermediate state when overriding a previous reassignment. Instead, the logic now jumps directly from the old reassignment to the new one and stops any replicas left in the dust. - Upon reassignment completion for a partition, the logic now *always* checks for a zk partition reassignment to be removed. This prevents any inconsistencies as a result of concurrent reassignments. - Api reassignments explicitly remove reassignment state from zk prior to beginning the new reassignment. I thought it was a little dangerous to allow both the zk reassignment and the api reassignment to exist at the same time.
2. @hachikuji Would you consider these a blocker for 2.4? cc @omkreddy

3. @viktorsomogyi Thanks for reviewing. I'll file a jira. Initially this was intended to be a minor fix, but it grew as I started digging into the logic. @ijuma I wouldn't call it a blocker. I think the patch makes the logic more robust and hopefully easier to follow, but there is not a specific known bug that it fixes.
4. Jenkins hit a couple OOMs before running. I built successfully against 2.11 and 2.12 locally. I will go ahead and merge.

github_pulls_reviews:

1. nit: Not sure if we need this? `updateReplicaAssignmentForPartition` already contains a similar statement
2. nit: could we have `topicPartition` here too?
3. nit: Could we use `topicPartition` as the parameter name for consistency please? Using `partition` might be a bit confusing.
4. nit: using `List.empty` might improve readability
5. nit: `Seq.empty` could be better for readability
6. This is an important side effect of the function, I think we should add this to the javadoc comment.
7. Turns out we didn't need the LeaderAndIsr here? I thought we needed it to prevent the leader from adding any replica outside the replica set back into ISR. I guess our very next action is to send a LeaderAndIsr in Phase A so it doesn't matter?
8. If the controller loses connectivity at this point (after updating ZK and before calling StopReplica), would we be left with extra replicas or would the leader kick them out?
9. So we would only remove it from ZK if the replicas are the same as the assignment. Unless I'm misreading, what happens here if: ``` 1. ZK call - reassign tp1 => [1,2,3] 2. API call - reassign tp1 => [3,4,5] ``` Would we ever remove the ZK persisted `[1,2,3]`?
10. Could we update the javadoc? or I guess move it to `onPartitionReassignment`
11. Do we handle this case anywhere? It would result in dubious requests send as we would enter phase B of `onPartitionReassignment` immediately
12. Curious as to why we don't fall back to the ZK-written assignment? We sometimes update ZK first which would make this a non-issue but what happens if we fail in `B3`? If I'm reading it right, we would have `RS=TRS` in memory while ZK would contain something else. If a user retries that reassignment, we would ignore the value in ZK and only retire the old TRS replicas.
13. Good call this surely doesn't warrant an `error` level
14. typo - `100`
15. Could we update the javadoc?
16. Ok. Let me remove this and try to improve the log message in `updateReplicaAssignmentForPartition`. I want it to include both the previous state and the updated state.
17. On second thought, perhaps I will just reduce the log level here.
18. It's a good question. I think we have this problem for replica removal in general even prior to KIP-455. The StopReplica requests are all sent asynchronously and we do not await their completion as we do with topic deletion. Generally our approach is pretty brittle. I think this will be fixed by KIP-500.
19. I was thinking it was unnecessary because we already get an epoch bump in A3, but in the case of a cancellation, we may not hit that case at all. So perhaps we still need it. It's a bit tough at the moment to see all of the epoch bumps because some of them are hidden beneath replica state changes. Let me look at this more carefully.
20. I changed the logic so that the controller context is only updated after we have updated the assignment state in zookeeper. Unless I missed something, there shouldn't be any need to revert to a previous state.
21. It seemed simpler not to optimize for this case and let it fall through the normal reassignment completion logic. I think the only cost is a round of UpdateMetadata requests. Maybe that's fine considering it's weird to do a no-op reassignment. What do you think?
22. Actually I added an overload `apply` for this case, so we can remove the adding and removing args.
23. API calls will cancel zk reassignments explicitly. My concern was actually the opposite: if a zk reassignment arrives while we are completing an API reassignment.
24. Ok, I think I convinced myself that it is unnecessary. Whether we go down the phase A path or phase B path, we will get an epoch bump. In phase B, the epoch will be bumped in `moveReassignedPartitionLeaderIfRequired`. Does that seem right?
25. Ok.
26. This is an existing issue. I am wondering why the ordering of the original replica set is not preserved? We store replica set as a list of broker ids.
27. ZkPartitionReassignment triggers maybeTriggerPartitionReassignment(). After initializePartitionReassignment() is called in initializeControllerContext(), onControllerFailover() calls

maybeResumeReassignments() which duplicates some of the logic in maybeTriggerPartitionReassignment().
28. We probably want to use different states btw ZkPartitionReassignment and ApiPartitionReassignment?
29. Is ReassignPartitionsZNode used?
30. Let me try to clarify this in the comment since I was also confused by it. I think the problem is specifically when the target replica set overlaps with the existing replica set. For example, suppose you have an initial assignment of [1, 2, 3] and you want to reassign to [3, 4, 2]. While the reassignment is in progress, the target replicas are listed first, so we would have [3, 4, 2, 1]. We can guarantee the desired order when the reassignment completes, but we have lost the order of the original assignment. There's not really an obvious solution to this problem that I can see without adding additional state.
31. +1 to that clarification, this was my thinking when I wrote that comment
32. I also traced it down - I think you're right.
33. Oh. Sounds good then
34. Doesn't this method only remove them if `tp == topicPartition && replicas == assignment.replicas`. Would `replicas == assignment.replicas` be true in the example I gave?
35. I don't mind. I always prefer simplicity over performance in non-essential paths and this seems like one
36. Well, `moveReassignedPartitionLeaderIfRequired` would bump up the leader epoch and send a LAIR wave too, right?
37. The new logic guarantees that the reassignment is stored in ZK before it is added to `partitionsBeingReassigned` - nice!
38. The cleanup phase referred to the `maybeRevertOngoingReassignment` method. Now that we don't have it, I think we need to remove this section. While these steps looks true, I think we at least want to specify that 1-5 are steps in between phase B
39. We now do this all the time. Could we move A1 and A2 out of Phase A? Otherwise I read it as if we expect A3 to always happen after A2 - but this isn't the case
40. I thought the intent of `ControllerState` was to capture a higher level breakdown of where the controller was spending its time. For example, `AlterPartitionReassignment` state is used both for the initial reassignment trigger and for ISR state changes affecting a reassigning partition. It might be useful to see a finer-grained breakdown based on the trigger, but I think practically speaking only one of them will be used in a given cluster.
41. Good point. I pushed an update to address this.
42. Not in B2->B6. So the opposite could happen - if we throw an exception in between the ZK update and the memory update, we would have stale things in memory.
43. The api reassignment cancels any active zk assignment for the same partition without respect to the target replicas. The check we are discussing only applies when an assignment has been completed.
44. True. I think I still prefer not to optimize for this case since it simplifies the handling. One (debatable) benefit is that we could use a no-op reassignment to force LeaderAndIsr propagation, which would be lighter than a controller move.
45. Agreed
46. I agree this is still a weak point. If we want to handle it, it would make more sense to do it in `onPartitionReassignment` since phase B is unlikely to execute in `maybeTriggerPartitionReassignment` unless it is a no-op move. Also, I think we would have to do more than just reset the assignment state. We probably need to figure out how to retry. The controller could track a failed reassignments collection, for example, which we could periodically retry. Since this is a pre-existing problem, I will file a JIRA to follow up with this.
47. Thanks. I've updated the doc to clarify the phases.
48. fyi: https://issues.apache.org/jira/browse/KAFKA-9099

jira_issues:

1. **summary:** Reassignment should be resilient to unexpected errors
   description: Reassignment changes typically involve both an update to the assignment state in zookeeper and an update to the in-memory representation of that state (in the ControllerContext). We can run into trouble when these states get inconsistent with each other, so the reassignment logic attempts to follow some rules to reduce the impact from this: * When creating a new reassignment, we update the state in zookeeper first before updating memory. Until the reassignment is known to be persisted, we do not begin executing any reassignment logic. * When completing a reassignment, all of the completion steps are executed before the state is updated in zookeeper. In the event of a failure, the new controller can retry reassignment completion. However, the current logic does not follow these rules strictly which can lead to

state inconsistencies in the case of an unexpected error. # When we override or cancel an existing assignment, we currently use an intermediate assignment state which is only reflected in memory. It is basically a mix of the previous assignment state and the overlapping parts of the new reassignment. The purpose of this is to shutdown unneeded replicas from the existing reassignment. Since the intermediate state is not persisted, a controller failure will revert to the old reassignment. Any exception which does not cause a controller failure will result in state divergence. # The target replicas of a reassignment are represented both in the existing assignment (PartitionReplicaAssignment) and in a separate context object (ReassignedPartitionContext). The reassignment context is updated before a reassignment has been accepted and persisted. The intent is to remove this context object in the event of a submission failure, but an unexpected error will leave it around. We can make reassignment more resilient to unexpected errors by using consistent update invariants. Specifically we can remove the intermediate assignment state and enforce the invariant that any active reassignment must be persisted before being reflected in memory. Additionally, we can make the assignment state the source of truth for the target replicas and eliminate the possibility of inconsistency. Doing so simplifies the reassignment logic and makes it more resilient.

jira_issues_comments:

1. hachikuji commented on pull request #7562: KAFKA-9089; Reassignment should be resilient to unexpected errors URL: https://github.com/apache/kafka/pull/7562 ---------- This is an automated message from the Apache Git Service. To respond to the message, please log on to GitHub and use the URL above to go to the specific comment. For queries about this service, please contact Infrastructure at: users@infra.apache.org