Pause Consumption on Realtime Segments

Sajjad Moradi and Subbu Subramaniam

Introduction

There have been multiple requests from the Pinot community to support pause/resume consumption of realtime segments. The request is to pause consumption while still allowing queries to process the current consuming segment, and then resume after some action is completed (e.g. debugging, or swapping the stream underneath, etc.). This feature is also helpful during testing. This document proposes a design in which pausing and resuming consumption are initiated by admins using two new REST endpoints on the Pinot Controller.

In Appendix A, we also address how the design solves the open source issues #6302, #7280, #6854, #6679, #7100, and #7039.

New Controller Endpoints

Two new endpoints on the Controller will be used for pause and resume scenarios[1]. Figure 1 shows the interaction between different components for the happy path for these two endpoints. How these endpoints handle edge cases and possible failure scenarios are described in the subsequent section.

[1] It may be useful to have a “status endpoint” or combine it with one of the existing status APIs we have.

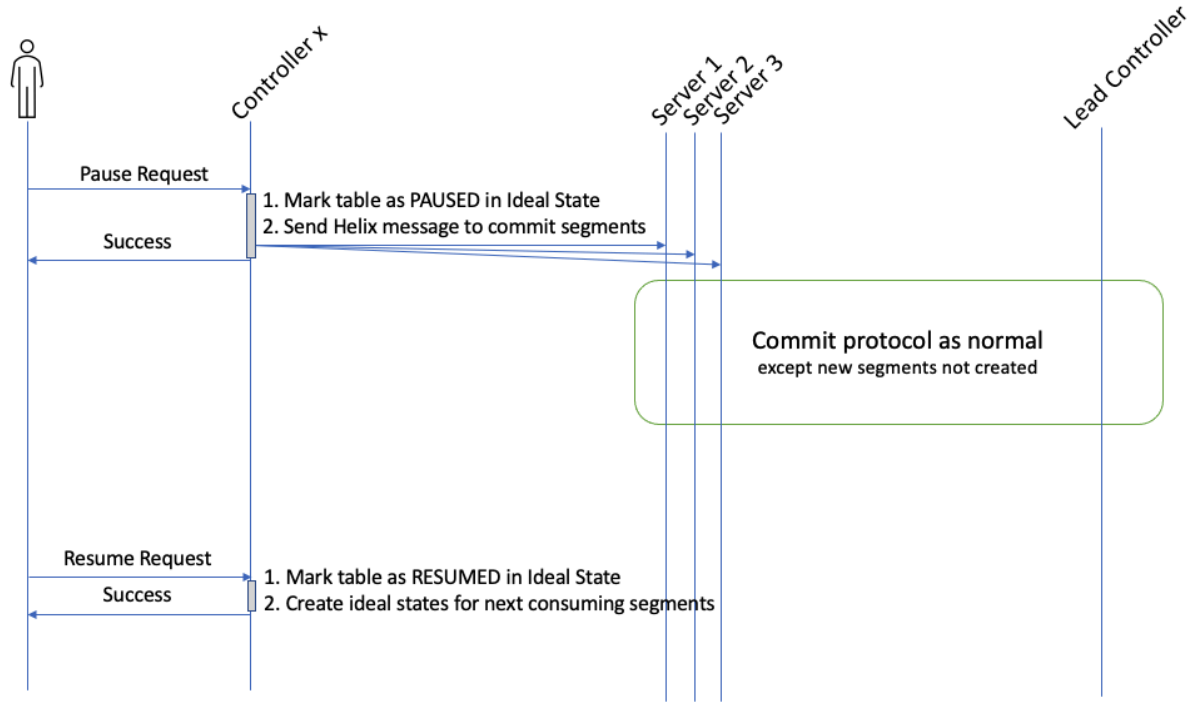

Figure 1. Pause and resume interactions

Pause Endpoint

On receiving a pause request, the Controller - which may not be the same controller that handles segment commits for the table - performs the following steps (this API is idempotent):

1. It marks the table as PAUSED in IdealState. For this purpose, we'll add a boolean field to "simpleFields" of the table's IdealState called "isTablePaused". The value true for this field means the table is PAUSED. Null value or false means table is not PAUSED.
2. It then sends a user defined Helix message to all servers hosting the given table to commit their segments.

The user-defined Helix message for commit has the following properties:

```
{
  "tableName": "tableXYZ_REALTIME",
  "segments": [tableXYZ__0__55__20220104T1810Z,
               tableXYZ__1__55__20220104T1811Z,
               tableXYZ__2__55__20220104T1812Z,
               tableXYZ__3__55__20220104T1812Z]
}
```

The "segments" field has the list of consuming segment names at the time the pause request was issued.

On the Server side, two modifications will be made:

1. As soon as the helix message to pause is received, the server notifies all the consuming segments to commit at whatever offsets they are on at the time.

2. When a consuming segment first comes into existence, it checks the ideal state for the PAUSED flag. If the flag is set, the server commits an empty segment.

Commit Protocol

All interactions between Servers and the lead Controller in the commit protocol remain the same except the parts on Controller where new consuming segment metadata is added to ZK and also new entries for all replicas of the next consuming segment are added to the ideal states. These two steps need to be modified. If the “isTablePaused” property is set to true in Ideal State, Controller skips these steps and there won’t be any new consuming segment for this partition in Ideal States and there won’t be any new segment ZK metadata either.

Resume Endpoint

Upon receiving a resume request, the Controller, in one Ideal State update operation, sets the “isTablePaused” flag to false and also creates new consuming segment entries. Creating the ideal state entries essentially kicks off the consumption for the partitions.

Alternative Designs

In this section, we go over alternative design proposals and also some improvements that can be omitted in the first implementation phase and added later on.

Use Zookeeper Watch

A minor deviation from the design proposed is to have the lead controller for the table set a zookeeper watch on the Ideal State, and trigger the helix messages off the zookeeper watch. . Figure 2 shows the details for this approach. Basically whenever the “isTablePaused” flag is set for a table in Ideal States, the lead controller for that table gets notified of the change by ZK and then it sends the Helix messages to servers to commit segments immediately.

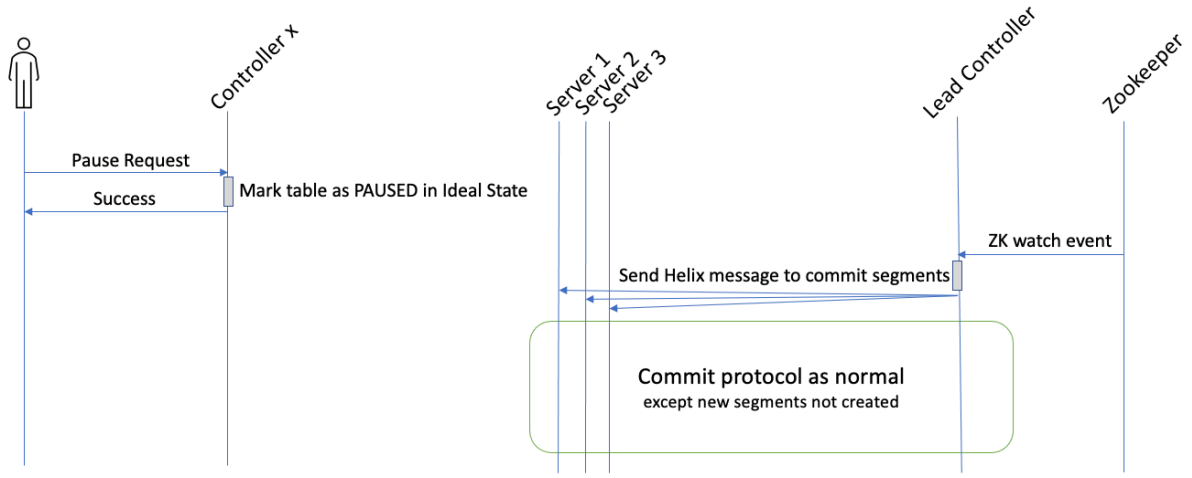

Figure 2. Zookeeper watch triggers commit messages

One advantage of this alternative approach is that the Ideal State is marked as paused - no matter via pause endpoint or manually updating the ZK - the lead controller will get notified and will send the commit messages for immediate pause.

The problems with this extension is that

1. The main problem is when a controller goes down and comes back up, upon establishing the ZK session, ZK sends the data for all the ZNodes for which watches are set. That will be a huge data for the Ideal States of all realtime tables and we've faced this kind of problem with too-many-watches before.
2. We need to add Zookeeper watches to all realtime tables, even the ones that are not involved in pause/resume operations.
3. The ideal state znode contains online/consuming/offline status for all segments and for any segment commit, there will be a watch event that we don't need to process for the purpose of pause/resume operations..
4. The controllers need to set a watch on the parent directory (IDEALSTATES) to get notified of any realtime tables added, so that they can set a watch on the appropriate one.
5. We need to add appropriate logic to add/remove watches when controller leadership for tables transfer.

We therefore believe it's not worth the cost and complexity of the alternative solution to support a case like Pause/Resume as per the requirements in the Issues cited. Further, the extension can be implemented as an add-on if we choose to add watches on the Ideal State at a later date for a new feature

Helix State Machine Modification

An alternative way to support the pause/resume feature is to modify the current Helix finite state machine. Last time we changed the state machine (to add a CONSUMING state), it was not

easy to come up with an upgrade plan (and order of upgrade) so that the change is transparent. Since we did not have live LLC use cases, it made it easier. Not so at this time, when we have multiple live installations across the world. Note that we also need a rollback plan in case things break (even if they break due to some other reason). This is the primary reason we do not favor this approach.

Having said that, let us try to evaluate what this approach may look like. The current state model looks as follows (simplified for realtime table only. The actual state model is more complex, since it also supports OFFLINE tables in the same state model.)

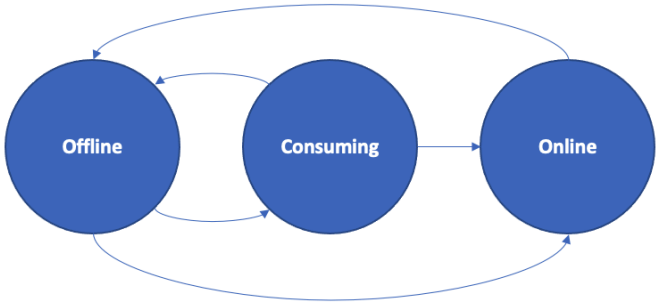

Figure 2. Current state machine for consuming segments

We have two options in this approach

1. We keep the current consuming segment(s) in memory
2. We complete the current consuming segment(s) but not create new ones

According to these choices, we may have multiple options to change the state machine. For the first option, we may want to add a PAUSED state between OFFLINE and CONSUMING states above. The idea is to go from CONSUMING to PAUSED and back. We see the following problems with this approach:

- When moving to PAUSED state, each replica of a CONSUMING segment can be at a different offset. This will cause inconsistent results depending on the replica queried.
    - In order to get away from this, we can replicate the segment commit protocol to implement a segment pause protocol, basically doing everything that segment commit does, except create a new segment, and allow each replica to reach the same offset as the other. Too complex, not worth it.
- Servers can be restarted during the pause. The idea here should be to come back to the same point the segment was paused at, which means we should record the pause offset, which means we should sync the pause offset across replicas – too complex.
- This does not address the use case in Issue 6302 (consume from different topics underneath after pausing)

So then let us move to option 2, where we complete the current consuming segment but not create a new one. The PAUSED state will be a transition state, basically triggering the servers to start the completion protocol. We will still need everything else that we have proposed – the znode to keep track of the status, etc. The resume will be exactly the same as what we have proposed – add new CONSUMING segments keeping the offsets sane.

Lastly, let us say the servers have already started the segment completion. And we get a request for pausing the stream. How does the server handle the state transition? We could not figure out an easy way to do this.

Partition-Level Pausing

The proposed design so far covers table-level pausing and resuming, meaning that all partitions of a table are paused or resumed. If a compelling reason comes up to support pausing only some partitions of a table, we can do so by adding a new field “pausedPartitions” in the Ideal State which is a list with the name of the paused partitions. This will be compatible with the boolean flag proposed in previous sections.

Appendix A - Related Github Issues

The following issues can directly or indirectly benefit from the pause/resume feature.

- **#6302 Support for pausing the realtime consumption without disabling the table.**
  The main request on this issue is to pause and resume the stream which is directly addressed by this design. There’s also another interesting use case where there’s a one-time large input and the user doesn’t want to setup offline flow. Data is in S3 and they want to stream it using Pulsar and when all the data is ingested they want to kill the topic so that there won’t be any useless pressure on their Pulsar cluster. The pause endpoint perfectly addresses this use case.

- **#7280 Add noDowntime option to reset**
  The force commit part of the pause endpoint can be reused in a separate API for resetting purposes.

- **#6854 Recover real-time consumption when consuming segment stuck**
  Pause/resume can be used in the case of schema evolution to recreate the consuming segment.

- **#7100 [question] Even if I recreate the kafka topic or modify the topic properties, I wonder how consumers can continue to do it.**
  Client issues pause request. Then the current consuming segments complete immediately and there won’t be any consuming segments which means there’s no kafka consumer open. Then the client can change the stream config properties including topic change and then issue the resume. The new consuming segments will pick up the new property from the table config.

- **#6679 Admin command to recover from deleted CONSUMING segment case**
  Part of the resume endpoint can be reused in a separate API to spin off new consuming segments.

- **#7039 More control of realtime segments - topic partitions offsets**
  Basically the same as #6679.