Item 199
git_comments:

1. segment1: 0 -> {0, 100}, 1 -> {1, 120} segment2: 2 -> {1, 120}, 3 -> {0, 100}
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. Remove the first segment
4. Add the first segment segment1: 0 -> {0, 100}, 1 -> {1, 120}, 2 -> {2, 100}
5. Add the second segment
6. Remove the original segment1
7. segment1: 0 -> {0, 100}, 1 -> {1, 120}, 2 -> {2, 100} segment2: 3 -> {0, 100}
8. segment2: 2 -> {0, 100}, 3 -> {0, 100}
9. Add the first segment
10. segment2: 2 -> {2, 120}, 3 -> {3, 80} new segment1: 0 -> {0, 100}, 1 -> {4, 120}
11. original segment1: 0 -> {0, 100}, 1 -> {4, 120} segment2: 2 -> {2, 120}, 3 -> {3, 80} new segment1: 0 -> {0, 100}, 1 -> {4, 120}
12. Update records from the second segment
13. Add 2 segments segment1: 0 -> {0, 100}, 1 -> {1, 100} segment2: 2 -> {0, 100}, 3 -> {0, 100}
14. segment1: 0 -> {0, 100}, 1 -> {4, 120} segment2: 2 -> {2, 120}, 3 -> {3, 80}
15. segment1: 0 -> {0, 100}, 1 -> {4, 120}, 2 -> {2, 100}
16. Replace (reload) the first segment
17. For upsert
18. * * Enables upsert for this segment. It should be called before the segment getting queried.
19. Skip pruning segments for upsert table because valid doc index is equivalent to a filter
20. Update the record location when getting a newer timestamp
21. The current record location has the same segment name
22. New primary key
23. The current record location is pointing to the new segment being loaded
24. Existing primary key Update the record location when getting a newer timestamp
25. The current record location is pointing to the old segment being replaced. This could happen when committing a consuming segment, or reloading a completed segment. Update the record location when the new timestamp is greater than or equal to the current timestamp. Update the record location when there is a tie because the record locations should point to the new segment instead of the old segment being replaced. Also, do not update the valid doc ids for the old segment because it has not been replaced yet.
26. * * Updates the upsert metadata for a new consumed record in the given consuming segment.
27. Check and remove to prevent removing the key that is just updated.
28. **comment:** Remove all the record locations that point to the valid doc ids of the removed segment.
    label: documentation
29. The current record location is pointing to a different segment

git_commits:

1. **summary:** Support reloading upsert table (#6167)
   message: Support reloading upsert table (#6167) Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.

github_issues:

1. **title:** implement upsert support on pinot
   body: Pinot is a distributed real-time OLAP engine that can provide second-level data freshness by ingesting kafka events and capacity to manage months of historical data load from various data sources such as HDFS, schemaless, etc. However, Pinot right now mostly functions as an append-only storage system. It doesn’t allow modify/delete of existing records with the exception of overriding all data within a time range with offline tables. This limits the applicability of pinot system due to a lot of use-cases requiring updates to their data due to the nature of their events or needs for data correction/backfill. In order to extend the capacity of pinot and serve more use-cases, we are going to implement the upsert features in Pinot which allows users to update existing records in Pinot tables with its kafka data input stream. Some initial requirements for the upsert projects: 1. Only support for full update to pinot event 2. Only support for Kafka-compatible queue ingestion model 3. Single pinot server/table can handle 10k/sec ingestion message rate 4. Each pinot server can handle 1 Billion records or 1TB storage 5. Ingestion latency overhead compared to non-upsert model < 1min 6. Query latency overhead compared to non-upsert model < 50% 7. Data retention < 2 weeks
   label: code-design
2. **title:** implement upsert support on pinot
   body: Pinot is a distributed real-time OLAP engine that can provide second-level data freshness by ingesting kafka events and capacity to manage months of historical data load from various data sources such as HDFS, schemaless, etc. However, Pinot right now mostly functions as an append-only storage system. It doesn’t allow modify/delete of existing records with the exception of overriding all data within a time range with offline tables. This limits the applicability of pinot system due to a lot of use-cases requiring updates to their data due to the nature of their events or needs for data correction/backfill. In order to extend the capacity of pinot and serve more use-cases, we are going to implement the upsert features in Pinot which allows users to update existing records in Pinot tables with its kafka data input stream. Some initial requirements for the upsert projects: 1. Only support for full update to pinot event 2. Only support for Kafka-compatible queue ingestion model 3. Single pinot server/table can handle 10k/sec ingestion message rate 4. Each pinot server can handle 1 Billion records or 1TB storage 5. Ingestion latency overhead compared to non-upsert model < 1min 6. Query latency overhead compared to non-upsert model < 50% 7. Data retention < 2 weeks
   label: documentation
3. **title:** implement upsert support on pinot
   body: Pinot is a distributed real-time OLAP engine that can provide second-level data freshness by ingesting kafka events and capacity to manage months of historical data load from various data sources such as HDFS, schemaless, etc. However, Pinot right now mostly functions as an append-only storage system. It doesn’t allow modify/delete of existing records with the exception of overriding all data within a time range with offline tables. This limits the applicability of pinot system due to a lot of use-cases requiring updates to their data due to the nature of their events or needs for data correction/backfill. In order to extend the capacity of pinot and serve more use-cases, we are going to implement the upsert features in Pinot which allows users to update existing records in Pinot tables with its kafka data input stream. Some initial requirements for the upsert projects: 1. Only support for full update to pinot event 2. Only support for Kafka-compatible queue ingestion model 3. Single pinot server/table can handle 10k/sec ingestion message rate 4. Each pinot server can handle 1 Billion records or 1TB storage 5. Ingestion latency overhead compared to non-upsert model < 1min 6. Query latency overhead compared to non-upsert model < 50% 7. Data retention < 2 weeks
   label: documentation
4. **title:** implement upsert support on pinot
   body: Pinot is a distributed real-time OLAP engine that can provide second-level data freshness by ingesting kafka events and capacity to manage months of historical data load from various data sources such as HDFS, schemaless, etc. However, Pinot right now mostly functions as an append-only storage system. It

doesn’t allow modify/delete of existing records with the exception of overriding all data within a time range with offline tables. This limits the applicability of pinot system due to a lot of use-cases requiring updates to their data due to the nature of their events or needs for data correction/backfill. In order to extend the capacity of pinot and serve more use-cases, we are going to implement the upsert features in Pinot which allows users to update existing records in Pinot tables with its kafka data input stream. Some initial requirements for the upsert projects: 1. Only support for full update to pinot event 2. Only support for Kafka-compatible queue ingestion model 3. Single pinot server/table can handle 10k/sec ingestion message rate 4. Each pinot server can handle 1 Billion records or 1TB storage 5. Ingestion latency overhead compared to non-upsert model < 1min 6. Query latency overhead compared to non-upsert model < 50% 7. Data retention < 2 weeks

5. **title:** implement upsert support on pinot
   body: Pinot is a distributed real-time OLAP engine that can provide second-level data freshness by ingesting kafka events and capacity to manage months of historical data load from various data sources such as HDFS, schemaless, etc. However, Pinot right now mostly functions as an append-only storage system. It doesn’t allow modify/delete of existing records with the exception of overriding all data within a time range with offline tables. This limits the applicability of pinot system due to a lot of use-cases requiring updates to their data due to the nature of their events or needs for data correction/backfill. In order to extend the capacity of pinot and serve more use-cases, we are going to implement the upsert features in Pinot which allows users to update existing records in Pinot tables with its kafka data input stream. Some initial requirements for the upsert projects: 1. Only support for full update to pinot event 2. Only support for Kafka-compatible queue ingestion model 3. Single pinot server/table can handle 10k/sec ingestion message rate 4. Each pinot server can handle 1 Billion records or 1TB storage 5. Ingestion latency overhead compared to non-upsert model < 1min 6. Query latency overhead compared to non-upsert model < 50% 7. Data retention < 2 weeks
6. **title:** implement upsert support on pinot
   body: Pinot is a distributed real-time OLAP engine that can provide second-level data freshness by ingesting kafka events and capacity to manage months of historical data load from various data sources such as HDFS, schemaless, etc. However, Pinot right now mostly functions as an append-only storage system. It doesn’t allow modify/delete of existing records with the exception of overriding all data within a time range with offline tables. This limits the applicability of pinot system due to a lot of use-cases requiring updates to their data due to the nature of their events or needs for data correction/backfill. In order to extend the capacity of pinot and serve more use-cases, we are going to implement the upsert features in Pinot which allows users to update existing records in Pinot tables with its kafka data input stream. Some initial requirements for the upsert projects: 1. Only support for full update to pinot event 2. Only support for Kafka-compatible queue ingestion model 3. Single pinot server/table can handle 10k/sec ingestion message rate 4. Each pinot server can handle 1 Billion records or 1TB storage 5. Ingestion latency overhead compared to non-upsert model < 1min 6. Query latency overhead compared to non-upsert model < 50% 7. Data retention < 2 weeks
7. **title:** implement upsert support on pinot
   body: Pinot is a distributed real-time OLAP engine that can provide second-level data freshness by ingesting kafka events and capacity to manage months of historical data load from various data sources such as HDFS, schemaless, etc. However, Pinot right now mostly functions as an append-only storage system. It doesn’t allow modify/delete of existing records with the exception of overriding all data within a time range with offline tables. This limits the applicability of pinot system due to a lot of use-cases requiring updates to their data due to the nature of their events or needs for data correction/backfill. In order to extend the capacity of pinot and serve more use-cases, we are going to implement the upsert features in Pinot which allows users to update existing records in Pinot tables with its kafka data input stream. Some initial requirements for the upsert projects: 1. Only support for full update to pinot event 2. Only support for Kafka-compatible queue ingestion model 3. Single pinot server/table can handle 10k/sec ingestion message rate 4. Each pinot server can handle 1 Billion records or 1TB storage 5. Ingestion latency overhead compared to non-upsert model < 1min 6. Query latency overhead compared to non-upsert model < 50% 7. Data retention < 2 weeks

github_issues_comments:

1. **body:** summary on 1st discussion of upsert design (May 15th): 1. @Jackie-Jiang points out that current design of rewriting queries for upsert table is problematic as this cause too much overhead in pinot query process with so many or-conditions. We should look into method to reduce the query overhead in upsert table 2. We should look into message delivery from coordinator service to pinot server. @Jackie-Jiang proposed methods on re-using existing download API on coordinator service to deliver messages from coordinator to server for unified API. 3. How to handle kafka topic partition change. @mcvsubbu suggested that we should look into how to handle accidental expand of kafka topic if that happens.
   label: code-design
2. **body:** @jamesyfshao can you add a pointer to your design doc in this issue? thanks
   label: documentation
3. **body:** > @jamesyfshao can you add a pointer to your design doc in this issue? thanks please use this design doc for now https://docs.google.com/document/d/1SFFir7ByxCff-aVYxQeTHpNhPXeP5q7P4g_6O2iNGgU/edit. We might have permission issue for new ppl as it is still WIP to keep the discussion more organized. Once it is more close to "ready" state I will put it on apache site
   label: documentation
4. It has been a while since the upsert work started. We'd like to reflect on the challenges encountered, share some learnings and also a [revisit on the upsert design](https://docs.google.com/document/d/1qljEMndPMxbbKtjlVn9mn2toz7Qrk0TGQsHLfI--7h8/edit#heading=h.lsfmyoyyxtgt).
5. Added the docs at https://docs.pinot.apache.org/basics/data-import/upsert

github_pulls:

1. **title:** Support reloading upsert table
   body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
2. **title:** Support reloading upsert table
   body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
3. **title:** Support reloading upsert table
   body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
   label: documentation
4. **title:** Support reloading upsert table
   body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
   label: code-design
5. **title:** Support reloading upsert table
   body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
6. **title:** Support reloading upsert table
   body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that

the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.

7. **title:** Support reloading upsert table
   body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
8. **title:** Support reloading upsert table
   body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
9. **title:** Support reloading upsert table
   body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
   label: code-design
10. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
    label: code-design
11. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
12. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
    label: documentation
13. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
    label: documentation
14. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
15. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
16. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
17. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
18. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
    label: code-design
19. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
    label: code-design
20. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
21. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
22. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.

23. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.
24. **title:** Support reloading upsert table
    body: ## Description Part of a series of PRs for #4261 Re-implement the `PartitionUpsertMetadataManager` to correctly handle the following 2 scenarios: 1. Reload the segment which replaces the upsert metadata of the existing segment 2. Manage the valid doc ids of the consuming segment within the manager so that the updates can be applied to the consuming segment One behavior change is that if 2 records have the same timestamp, the second one won't replace the first one in order to reduce the number of updates. Add `PartitionUpsertMetadataManagerTest` to verify the functionalities.

github_pulls_comments:

1. # [Codecov](https://codecov.io/gh/apache/incubator-pinot/pull/6167?src=pr&el=h1) Report > Merging [#6167](https://codecov.io/gh/apache/incubator-pinot/pull/6167?src=pr&el=desc) into [master](https://codecov.io/gh/apache/incubator-pinot/commit/1beaab59b73f26c4e35f3b9bc856b03806cddf5a?el=desc) will **increase** coverage by `6.82%`. > The diff coverage is `60.00%`. [![Impacted file tree graph](https://codecov.io/gh/apache/incubator-pinot/pull/6167/graphs/tree.svg?width=650&height=150&src=pr&token=4ibza2ugkz)](https://codecov.io/gh/apache/incubator-pinot/pull/6167?src=pr&el=tree) ```diff @@ Coverage Diff @@ ## master #6167 +/- ## == + Coverage 66.44% 73.27% +6.82% == Files 1075 1236 +161 Lines 54773 58436 +3663 Branches 8168 8653 +485 == + Hits 36396 42819 +6423 + Misses 15700 12799 -2901 - Partials 2677 2818 +141 ``` | Flag | Coverage Δ | | |---|---|---| | #integration | `46.36% <49.14%> (?)` | | | #unittests | `64.12% <38.09%> (?)` | | Flags with carried forward coverage won't be shown. [Click here](https://docs.codecov.io/docs/carryforward-flags#carryforward-flags-in-the-pull-request-comment) to find out more. | [Impacted Files](https://codecov.io/gh/apache/incubator-pinot/pull/6167?src=pr&el=tree) | Coverage Δ | | |---|---|---| | [...ot/broker/broker/AllowAllAccessControlFactory.java](https://codecov.io/gh/apache/incubator-pinot/pull/6167/diff?src=pr&el=tree#diff-cGlub3QtYnJva2VyL3NyYy9tYWluL2phdmEvb3JnL2FwYWNoZS9waW5vdC9icm9rZXIvYnJva2VyL0FsbG93QWxsQWNjZXNzQ29udHJvbEZhY3Rvcnkuam
   | `100.00% <ø> (ø)` | | | [.../helix/BrokerUserDefinedMessageHandlerFactory.java](https://codecov.io/gh/apache/incubator-pinot/pull/6167/diff?src=pr&el=tree#diff-cGlub3QtYnJva2VyL3NyYy9tYWluL2phdmEvb3JnL2FwYWNoZS9waW5vdC9icm9rZXIvYnJva2VyL2hlbGl4L0Jyb2tlclVzZXJEZWZpbmVkTWVzc2FnZUhhb
   | `52.83% <0.00%> (-13.84%)` | :arrow_down: | | [...ava/org/apache/pinot/client/AbstractResultSet.java](https://codecov.io/gh/apache/incubator-pinot/pull/6167/diff?src=pr&el=tree#diff-cGlub3QtY2xpZW50cy9waW5vdC1qYXZhLWNsaWVudC9zcmMvbWFpbi9qYXZhL29yZy9hcGFjaGUvcGlub3QvY2xpZW50L0Fic3RyYWN0UmVzdWx0U2
   | `53.33% <0.00%> (-3.81%)` | :arrow_down: | | [.../main/java/org/apache/pinot/client/Connection.java](https://codecov.io/gh/apache/incubator-pinot/pull/6167/diff?src=pr&el=tree#diff-cGlub3QtY2xpZW50cy9waW5vdC1qYXZhLWNsaWVudC9zcmMvbWFpbi9qYXZhL29yZy9hcGFjaGUvcGlub3QvY2xpZW50L0Nvbm5lY3Rpb24uamF2YQ==
   | `44.44% <0.00%> (-4.40%)` | :arrow_down: | | [.../org/apache/pinot/client/ResultTableResultSet.java](https://codecov.io/gh/apache/incubator-pinot/pull/6167/diff?src=pr&el=tree#diff-cGlub3QtY2xpZW50cy9waW5vdC1qYXZhLWNsaWVudC9zcmMvbWFpbi9qYXZhL29yZy9hcGFjaGUvcGlub3QvY2xpZW50L1Jlc3VsdFRhYmxlUmVzdWx0
   | `24.00% <0.00%> (-10.29%)` | :arrow_down: | | [...not/common/assignment/InstancePartitionsUtils.java](https://codecov.io/gh/apache/incubator-pinot/pull/6167/diff?src=pr&el=tree#diff-cGlub3QtY29tbW9uL3NyYy9tYWluL2phdmEvb3JnL2FwYWNoZS9waW5vdC9jb21tb24vYXNzaWdubWVudC9JbnN0YW5jZVBhcnRpdGlvbnNVdGlscy5qYX
   | `78.57% <ø> (+5.40%)` | :arrow_up: | | [.../apache/pinot/common/exception/QueryException.java](https://codecov.io/gh/apache/incubator-pinot/pull/6167/diff?src=pr&el=tree#diff-cGlub3QtY29tbW9uL3NyYy9tYWluL2phdmEvb3JnL2FwYWNoZS9waW5vdC9jb21tb24vZXhjZXB0aW9uL1F1ZXJ5RXhjZXB0aW9uLmphdmE=) | `90.27% <ø> (+5.55%)` | :arrow_up: | | [...pinot/common/function/AggregationFunctionType.java](https://codecov.io/gh/apache/incubator-pinot/pull/6167/diff?src=pr&el=tree#diff-cGlub3QtY29tbW9uL3NyYy9tYWluL2phdmEvb3JnL2FwYWNoZS9waW5vdC9jb21tb24vZnVuY3Rpb24vQWdncmVnYXRpb25GdW5jdGlvblR5cGUuamF2Y
   | `98.27% <ø> (-1.73%)` | :arrow_down: | | [.../pinot/common/function/DateTimePatternHandler.java](https://codecov.io/gh/apache/incubator-pinot/pull/6167/diff?src=pr&el=tree#diff-cGlub3QtY29tbW9uL3NyYy9tYWluL2phdmEvb3JnL2FwYWNoZS9waW5vdC9jb21tb24vZnVuY3Rpb24vRGF0ZVRpbWVQYXR0ZXJuSGFuZGxlci5qYXZh)
   | `83.33% <ø> (ø)` | | | [...ot/common/function/FunctionDefinitionRegistry.java](https://codecov.io/gh/apache/incubator-pinot/pull/6167/diff?src=pr&el=tree#diff-cGlub3QtY29tbW9uL3NyYy9tYWluL2phdmEvb3JnL2FwYWNoZS9waW5vdC9jb21tb24vZnVuY3Rpb24vRnVuY3Rpb25EZWZpbml0aW9uUmVnaXN0cnkuam
   | `88.88% <ø> (+44.44%)` | :arrow_up: | | ... and [1003 more](https://codecov.io/gh/apache/incubator-pinot/pull/6167/diff?src=pr&el=tree-more) | | ------ [Continue to review full report at Codecov](https://codecov.io/gh/apache/incubator-pinot/pull/6167?src=pr&el=continue). > **Legend** - [Click here to learn more](https://docs.codecov.io/docs/codecov-delta) > `Δ = absolute <relative> (impact)`, `ø = not affected`, `? = missing data` > Powered by [Codecov](https://codecov.io/gh/apache/incubator-pinot/pull/6167?src=pr&el=footer). Last update [1bf5d02...392dfed](https://codecov.io/gh/apache/incubator-pinot/pull/6167?src=pr&el=lastupdated). Read the [comment docs](https://docs.codecov.io/docs/pull-request-comments).

github_pulls_reviews:

1. **body:** Not sure if this old segment check is necessary: - since the old segment will be replaced, it shall be safe to update the valid doc, since it will be gone anyway? - if so, then the handling is identical to the branch above, and therefore can be merged?
   label: documentation
2. **body:** wrap this in the else branch for better readability.
   label: code-design
3. Is it possible that this immutable segment is queried before the `enableUpsert ` is invoked? If so, `_validDocIndex ` will be null and confuse the query plan
4. does this check the case that a replaced segment shall not remove the keys of the newly loaded? Perhaps we shall consider a state of tracking the current segmentImpl (and its corresponding `validDocIds`) for a segment name?
5. it's worth explaining this a bit on which data structures won't be reflected.
6. how is consuming segment related?
7. **body:** not sure if `ThreadSafeMutableRoaringBitmap ` is the best identifier of the containing segment. Perhaps the segmentImpl itself, in case `ThreadSafeMutableRoaringBitmap ` itself may be replaced?
   label: code-design
8. **body:** nit: I think it's preferred to enable it as early as possible (i.e in the constructor), we know this segment will be an upsert one.
   label: code-design
9. shall we include the removal as part of the replace? the removal of the old shall be after the addition of the new?
10. **body:** When we commit a consuming segment, or reload a completed segment, the data will be identical (could be re-ordered). If we update the valid doc, before we replace the old segment in the data manager, all the docs in the old segment will be invalidated. Even though it can recover after the segment is replaced, we will observe data loss before that.
    label: documentation
11. **body:** No, the `enableUpsert` is called before adding the segment to the data manager. Will add more javadoc stating that
    label: documentation
12. Done
13. Good point, done
14. Just some explanation on why we don't need to call this method for the consuming segment when it is destroyed
15. Yes, it removes the entry only when the reference of the valid doc ids are the same. It won't touch the upsert metadata for other segments.
16. **body:** I don't want to put too many unrelated info here. If we decide to use another data structure, we can change the constructor. This class is just a wrapper, not an interface, so changing constructor should be fine.
    label: code-design

17. **body:** I moved this method out of the constructor because the segment loader doesn't need to know whether the segment has upsert enabled or not, and upsert metadata is updated after the segment is loaded. Decoupling upsert from the segment loader help simplify the code path for the loading part, and keep all the upsert handling at the same place: `RealtimeTableDataManager.addSegment(ImmutableSegment immutableSegment)`
    label: code-design
18. Good point, added
19. ah I see. So the window could be until the segment seal. Makes sense.
20. sgtm
21. okay, that's fair

jira_issues:

jira_issues_comments: