Author: Xin Gao X G gaoxin@uber.com
Status: Done
Pinot Github Issue https://github.com/apache/pinot/issues/13780
Pinot Github PR https://github.com/apache/pinot/pull/13790

Justification

Pinot nowadays only supports realtime table ingested from one single source stream, e.g. one Kafka topic from a Kafka cluster. And inside the table manager, the internal segment partition concept is hard coupled with the stream's partition. For example, if Kafka topic has 8 partitions, then Pinot table segments are also partitioned by 8, and each segment is consuming from the Kafka topic partition with the exact same partition id.
This is a workable and simple design which could fit most straightforward use cases. But it also imposes the flexibilities on ingestions.
In reality, users may produce data of the same subject to different Kafka topics and ingest to a single Pinot table (with the same Schema) to do centralized analysis. There was one Pinot open issue asking for the feature #5647. Other OLAP technologies, e.g. Clickhouse and Druid, are developing or have developed similar features like https://clickhouse.com/docs/en/engines/table-engines/integrations/kafka and apache/druid#14424.

Proposal

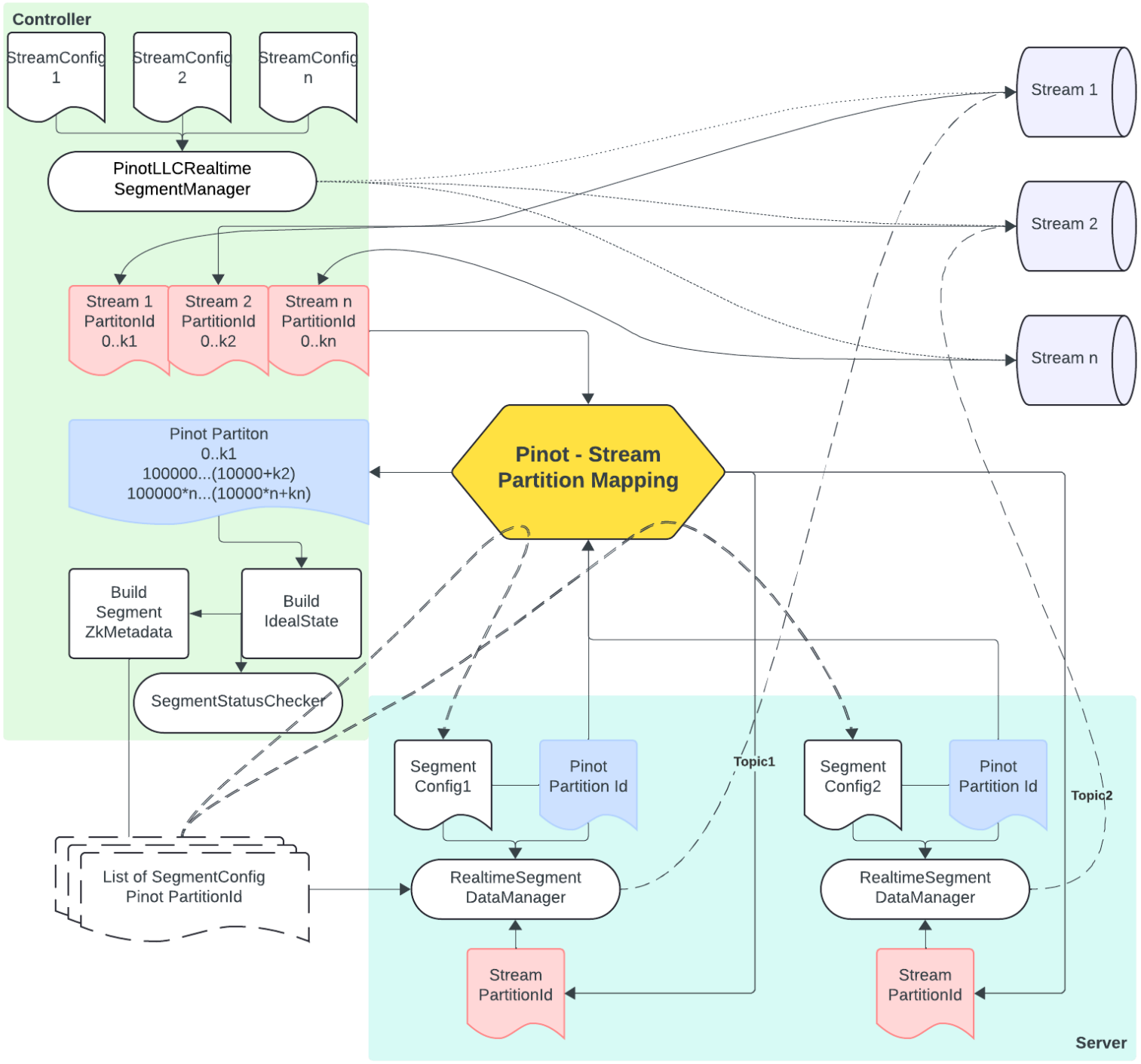
The implementation strategy should consider loosening the coupling of the partition concept between stream and Pinot. Theoretically, stream and OLAP db are two independent infra and storages. They should have their own partition strategies instead of having hard dependencies on the other. Pinot segment partition is only directly used for segment management. The data consumption of each segment partition should not be hardly coupled with the stream's partition. The abstraction layer could be built in between to manage the mapping.

Based on the current Pinot architecture, it is possible to add the feature with following features and constraints:

- Ingests from multiple stream topics and forms the same Pinot table.
- Different stream topics could be with different number of partitions, and even different data format (json, avro, protobuf, etc) meaning Pinot tables should be able to use different decoders to decode data from different tables accordingly.
- Same transformation and indexing strategy is applied to the decoded data from different topics. This limitation is due to the TableConfig structure we are defining, could be resolved if some major TableConfig refactor is done. Even with this limitation, transformation could be easily done by using existing dynamic transformation features like SchemaConformingTransformer introduced in Add SchemaConformingTransformerV2 to enhance text search abilities #12788.
- Only supports LLC.
- Table schema evolution, stream partition number expansion and auto catch-up, instance assignment strategies need to have same support without regressions.
- In the short term, we do not consider adding or removing topics from the stream topics list.

Design

Architecture



The plot above shows the design of partition related footprint.

- TableConfig allows multiple StreamConfigs maps.
- Stream PartitionId and Pinot PartitionId will be 2 separate concepts highlighted in red and blue boxes.
- A new partition mapping service will be introduced with following functionalities
    - Convert Stream PartitionId to Pinot PartitionId.
    - Convert Pinot PartitionId to Stream PartitionId.
    - Extract a specific SegmentConfig from List<StreamConfig> inside TableConfig with the help of Segment's Pinot PartitionId.
- Controller would
    - Fetch partition info from all stream topics and convert multiple lists of Stream PartitionIds to one Pinot PartitionId list.
    - Manage all segment status with Pinot PartitionId in the same way as before.
- Server would
    - Report and sync segment status with Pinot PartitionId same as before.
    - Consume data with the extracted single SegmentConfig and Stream PartitionId.

The design would keep segment commitment, status checking, data consumption logics same as before and achieve multi-topics ingestion by introducing a simple partition mapping component.

Pinot - Stream Partition Mapping

The partition mapping needs to follow the principle of

- Consistent reversible transformation.
- Could support interface
  SegmentConfig segmentStreamConfig = getSegmentConfigFromSegmentConfigs(List<SegmentConfig> segmentConfigs, int PinotPartitionId)
- Could tolerate partition expansion on any stream topics.
  - Data ingestion would be able to catch up and consume the newly created segments in the same way as today.
  - New partitions should not affect existing partitions of other unchanged topics.
- Could use the same existing instance partition assignment strategies.

Hence, a simple but efficient solution could be adding a static large offset (e.g. 10000) to each topic's partitions. For instance, assuming we want to ingest from 3 topics

- topic0: 8 partitions
- topic1: 4 partitions
- topic2: 16 partitions

Then Pinot partitionIds would be (0, 1, 2 … 7, 10000, 10001, 10002, 10003, 20000, 20001 … 20015), so that

- pinotPartitionId = 10000 * topicIndex + streamPartitionId
- streamPartitionId = pinotPartitionId % 10000
- topicIndex = pinotPartitionId / 10000

As the static offset is far larger than the actual partition number in the real world (~512), partition expansion in any topic would be transparent. E.g. topic1 expands to 8 partitions, then the controller will create new pinotPartitionIds (10004,10005,10006,10007) and assign new segment threads to consume as is.

Upsert Support (Not supported for now)

Upsert for multi-topics tables would be more complicated. We need first to have a clear definition and restrictions on the support scope.

- Can input streams be with different types? (e.g. Kafka & Kinesis)
- Can input streams be with different schemas? If so, can they be completely different or require some common parts?
- Can input streams be partitioned by different keys?
- Can input streams have different numbers of partitions? Can the number of partitions expand as will?
- Other common characteristics required for upstreams?

Instance and Segment Assignment

Segment assignment is a Pinot internal data distribution algorithm. It is not and should not be directly affected by stream partitions. Therefore, only the Pinot partitionId (instead of stream partition id) takes effect during the process.
There are 2 steps, instance selection and segment assignment.
The instance selection is based on table tenant and instanceAssignmentConfigMap. It does not rely on the partition status of the table.
As of segment assignment, the interface
assignConsumingSegment(int segmentPartitionId, InstancePartitions instancePartitions)
is taking PinotPartitionId as input and pick the instance in the way like
(segmentPartitionId * _replication + replicaId) % numInstances

With the Pinot-Stream Partition Mapping introduced as above, segmentPartitionId will remain as int and the same function could be used with similar effect without any change. For the same example,
Topic 0 segments will be assigned as (segmentPartitionId * _replication + replicaId) % numInstances
Topic 1 segments will be assigned as (10000 * _replication + segmentPartitionId * _replication + replicaId) % numInstances = (10000 * _replication) % numInstances + (segmentPartitionId * _replication + replicaId) % numInstances = **constant_offset** + (segmentPartitionId * _replication + replicaId) % numInstances

Topic 2 segments will be assigned as (20000 * _replication + segmentPartitionId * _replication + replicaId) % numInstances = **constant_offset * 2** + (segmentPartitionId * _replication + replicaId) % numInstances
Each stream/topic will use the same algorithm to do segment assignment with a shift on constant offset (e.g. assign to instance 0-7 vs 2-9).
Even if one of the topic's partition numbers gets expanded, only that topic's segment assignment gets affected. Other topic's assignment remains unchanged.

Other improvements

With this feature, it could also enhances ingestion performance and solves the issue like #13319 to have multiple segment partitions consuming from the same topic partition.
n:1 mapping for parallel consumption on one topic partition would be possible. E.g. PinotPartition 1,1001,2001 consumes StreamPartition 1 where PinotPartition 1 consumes offset % 3 ==0, 1001 consumes offset % 3 =1, etc. Same decoding, transformation, indexing and offset control logics could remain and achieve high ingestion speed regardless of the number of partitions on Kafka.

Reference

Multi-Stream Consumption in Pinot from LinkedIn (Proposal from this doc and the one from LinkedIn are designed and implemented independently)