KIP-315: Stream Join Sticky Assignor

- Status
- Motivation
- Public Interfaces
- Proposed Changes
    - Assignment Strategy
- Compatibility, Deprecation, and Migration Plan
- Rejected Alternatives

Status

Current state: *Discussion*

Discussion thread: TBD

JIRA: TBD

Example Code: https://github.com/MikeFreyberger/sticky-join-assignor

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

A popular use case for KafkaConsumers is performing a stream to stream join. The main requirement for the partition assignment is that all of the topics for a given partition must go to the same KafkaConsumer.

A canonical example is Ad Monetization where you have a impressions topic and a clicks topics. Each topic would have the same key (some sort of auction ID). Let's assume the topic is broken into 10 partitions, and there are 4 KafkaConsumers initially working on performing the stream to stream join.

An invalid assignment would be:

Consumer	Impression Topic Partitions Assigned	Click Topic Partitions Assigned
A	0,4,8	2,6
B	1,5,9	3,7
C	2,6	0,4,8
D	3,7	1,5,9

In this case we cannot perform a stream to stream join because a given partition is not uniquely owned by a single consumer. This assignment would be produced by the Round Robin Assignment.

A valid assignment would be:

Consumer	Impression Topic Partitions Assigned	Click Topic Partitions Assigned
A	0,1,2	0,1,2
B	3,4,5	3,4,5
C	6,7	6,7
D	8,9	8,9

In this case, Consumer A is responsible for all of partition 0,1, and 2 across all topics. Similarly, Consumer B is responsible for partition 3,4, and 5 across all topics. This is the assignment that would be produced by the Range Assignor.

While the Range Assignor produces a valid assignment for producing streaming joins, it does not remember the previous assignment during a consumer rebalance. So, when a consumer joins or leaves the group, the assignment will be completely re shuffled.

This is assignment if Consumer D leaves the group:

Consumer	Impression Topic Partitions Assigned	Click Topic Partitions Assigned
A	0,1,2,3	0,1,2,3
B	4,5,6	4,5,6
C	7,8,9	7,8,9

Partition 3 moved from Consumer B to Consumer A.

Partition 6 moved from Consumer C to Consumer B.

Partitions 8,9, which were unowned after Consumer D left, were assigned to Consumer C.

In this case, the movement of Partition 3 and 6 was unnecessary.

The assignor should try to minimize the movement of partitions when performing streaming joins. So a better assignment would have been:

Consumer	Impression Topic Partitions Assigned	Click Topic Partitions Assigned
A	0,1,2,9	0,1,2,9
B	3,4,5	3,4,5
C	6,7,8	6,7,8

In this case Partitions 8 and 9 are assigned to new consumers because they are unowned after D left the group. No other partitions have their assignment changed.

When performing stream to stream joins, the consumers typically maintain local state. Therefore, on partition assignment the state must be rebuilt by consuming a changelog, and on partition revocation, the state must be dropped. These operations can be expensive and slow, and should be minimized.

This is the same motivation behind the StickyAssignor that was added as a part of KIP-54. The main difference is that the StickyAssignor is a round robin assignment at its core, which violates the assignment requirements for doing a streaming join. This assignor will be sticky and will make sure to assign a particular partition number across all topics to the same consumer'.

Public Interfaces

There will be a new partition.assignment.strategy option*: StickyStreamJoinAssignor*

Proposed Changes

This assignment strategy should only be used if:

1. All consumers in the consumer group are subscribing to the same set of topics
2. All topics have the same number of partitions and the partitioning scheme is the same across all topics.

These expectations can be violated when:

1. A topic is being added or removed from the join
2. The partition count is increasing

The assignment strategy will handle these violations, but it will handle them in such a way that it assumes the initial expectations will be met again in a future rebalance.

Assignment Strategy

The assignment strategy is trying to assign same the partition across all topics to the same consumer. So the assignment strategy is focused on assigning partitions and then transforms those into TopicPartition assignments at the end.

1. **The assignor determines the number of partitions that must be assigned**. This is done by taking the minimum of the partition count across all topics being subscribed to in the group. During the normal case the partition count will be the same across all topics, so taking the minimum will have no affect. During edge cases, by taking the minimum certain TopicPartitions will go unassigned.
    a. Partitions being added to topics
        i. In this case the assumption is that the new partitions will not have data that is joinable until all of the topics have their partition count increase.
    b. Partitions being removed from topics
        i. This isn't safe and will most likely lead to data loss. This should not happen.
2. **The assignor determines the previous assignment.**
    a. Deserialize the user data that is passed in the subscription. The subscription contains all of the information that was given to the consumer on assignment.
        i. On assignment the consumer will store a HashSet<Integer> of partitions that were assigned to it. The assignment will also contain an epoch number of the rebalance (this is stored in the Assignment user data). This is necessary to protect against zombies. Therefore, the subscription will contain a list of partitions previously owned and the epoch number of the previous assignment (this is placed in the Subscription user data).
    b. Build a map of ConsumerList of Integers of previous ownership.
        i. This will only include partitions that are eligible for assignment.
        ii. This will prefer a previous assignment with a higher epoch. If Consumer A and Consumer B claim to have previously owned partition 0, it will pick which had true ownership by looking at the epoch number of Consumer A and Consumer B.
3. **Determine the unowned partitions.** Any partition that is not in the union of the list of previously owned partitions and less than the number of eligible partitions.
4. **Perform the new assignment**.
    a. Each consumer will select a partition 1 by 1 until all partitions have been selected. We will prefer a consumer who has a previously owned partition to perform the selection, while still being fair. The consumer will select a partition according to these rules:
        i. If there is a partition that they previously owned and no one has selected during this rebalance, select that partition.
        ii. If there are any unowned partitions, select that partition.
        iii. "Steal" a partition from a consumer that previously owned a partition. Stealing is ensured to be fair and stealing will begin with the consumer that had the longest list of previously owned partitions.
5. **Transform the assignment from Partitions to TopicPartitions.** After the assignment stage all consumers will have a list of partitions that are assigned to them. This partition list will be transformed into a TopicPartition list by looping through all of the topics in their subscription. If all subscriptions from all consumers have the same list of topics, all TopicPartitions will be assigned. During the edge case where not all subscriptions have the same list of topics, some TopicPartitions will go unassigned.
    a. If a particular Topic X has 100 partitions is only subscribed to by Consumer A which is assigned partitions 0-10, then Topic X partitions 11-99 would go un assigned.
        i. This allows for testing a new topic that is added to the join. So, if 1 consumer in the group is updated to subscribe to the new topic, it will not be overloaded by receiving the entire topic. It will only receive the portion of the topic that is relevant for performing the join.
6. **Determine the epoch.** The next epoch is the maximum epoch received from the all of the subscriptions + 1.
7. **Create the Assignment**. The assignment will include the epoch as a part of the Assignment user data.

Compatibility, Deprecation, and Migration Plan

This is only adding a new feature that is be default not enabled. No migration plan required.

Rejected Alternatives

None