KIP-341: Update Sticky Assignor's User Data Protocol

Status

Current state: *Accepted*

Discussion thread: *here*

Vote thread: *here*

JIRA: ***KAFKA-7026*** *- Getting issue details...* ***STATUS***

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

Motivation

Among the out-of-the-box consumer group partition assignment strategies, Sticky Assignor is the only one that is stateful. This is because in this assignment strategy one main goal is to preserve a consumer's assigned partitions as much as possible through rebalances; and for that each consumer reports its current assignment state to the leader during a rebalance. The leader then starts with the reported current assignments from all consumers in the group, and generates their new assignments with the goal of fairness and stickiness, in that particular order. According to the issue reported in KAFKA-7026 and the discussion that followed in the corresponding PR, there are rare cases where the current implementation could lead to assigning the same partition to multiple consumers, breaking the consumer group guarantee. Examples of how it breaks the guarantee exist in the PR discussion, but to summarize, if a consumer briefly leaves the group and then rejoin the group (thinking it has held onto its previous assignment while in reality its partitions were redistributed to other consumers as a result of it leaving the group), it will report its previous assignment to the leader and the leader would not have a way of detecting that the reported assignment is no longer valid. This reported stale assignment can easily conflict with reported (valid) assignments of other consumers and there will be a chance that the same partition will stay assigned to the consumers that reported it as part of their assignment, leading to a duplication. Note that this scenario would be a rare, and one way of reproducing it is running a consumer in debug mode and pausing it for long enough that causes a rebalance.

Public Interfaces

This is the existing user data protocol of Sticky Assignor.

```
User Data (Version: 0) => [previous_assignment]
  previous_assignment => topic [partitions]
    topic => STRING
    partitions => partition
      partition => INT32
```

Following each rebalance, the group leader sends each consumer a list of <topic, partitions> as its current partition assignment. The consumer sends that list back to the leader when the next rebalance starts.

Proposed Changes

In order to have an indication of how fresh a reported assignment by each consumer is the following protocol is suggested.

```
User Data (Version: 1) => [previous_assignment] generation
  previous_assignment => topic [partitions]
    topic => STRING
    partitions => partition
      partition => INT32
  generation => INT32
```

The only addition is a numeric generation marker that increases during each improvement. This lets the leader detect the highest generation and ignore a consumer's reported assignment (if necessary) when that assignment belongs to older generations.

Generation Marker

Consumer group generation will be used for the new user data field. On top of addressing the cons described below in Rejected Alternatives section for a local generation marker, this option also makes it possible for other stateful assignors to use consumer group generation.

This generation is not currently exposed to any of partition assignors. It will be exposed as part of the `assignor.onAssignment(...)` call in `ConsumerCoordinator.java`:

ConsumerCoordinator::onJoinComplete

```
// current code
...
assignor.onAssignment(assignment);
...

// revised code
...
assignor.onAssignment(assignment, generation);
...
```

As a result of above change to the `assignor.onAssignment(...)` call, a new method is introduced in the interface `PartitionAssignor`:

PartitionAssignor

```
default void onAssignment(Assignment assignment, int generation) {
    onAssignment(assignment);
}
```

The existing `onAssignment(...)` method will remain to support classes that already implement this interface.

It is expected that classes that implement this interface implement at least one of the two `assign(...)` methods. A default implementation is provided for this methods to keep assignors that do not make use of group generation (e.g. `RangeAssignor` and `RoundRobinAssignor` classes) intact.

Note: During the implementation of this KIP there was a back and forth on whether this generation argument should be an optional argument (i.e. `Optional<Integer>`). It turned out that the only edge case where the optional argument would be useful is when an old client makes use of the updated sticky assignor. It was decided that this supposedly rare case would not worth the complexity and the noise that comes with the optional type.

Compatibility, Deprecation, and Migration Plan

The new user data protocol can be implemented in a way that supports backward and forward compatibility.

- Backward compatibility: If the leader uses the new protocol it first tries to deserialize reported assignments of consumers using the new protocol. If that does not work it falls back to using the old protocol assuming a default (e.g. -1) generation number for the assignment.
- Forward compatibility: If the leader uses the old protocol it deserializes only the array portion of the reported assignment byte array ignoring the potential integer that could be following. Generation markers reported by new clients will all be ignored in this case.

Rejected Alternatives

- Using a local generation marker within the scope of Sticky Assignor: This option was voted against because 1) it could cause confusion with the consumer group generation; 2) it could reset to 0 from time to time (if no consumer in the group has a previous assignment); 3) there are corner cases where it may not lead to the expected optimum assignment (example)