KIP-792: Add "generation" field into consumer protocol

Status

Current state: *"Adopted"*

Discussion thread: *here*

***Vote thread**: here*

JIRA: *here*

Motivation

In KIP-429: Kafka Consumer Incremental Rebalance Protocol, we proposed a new version of consumer protocol for cooperative rebalance (a.k.a incremental rebalance protocol). It adds a new field "ownedPartition" into the "subscription" and "assignment" data, and one of the purpose of the new "ownedPartition" field is for the assignors and to do "sticky" assignment dependent on the "ownedPartitions". Like in the "*CooperativeStickyAssignor and custom COOPERATIVE Assignors*" section of KIP-429, it said:

The assignor can leverage the new ownedPartitions field that the Subscription has been augmented with in order to determine the previous assignment. Note that "stickiness" is important for the cooperative protocol to be effective, as in the limit that the new assignment is totally different than the previous one then the cooperative protocol just reduces to the old eager protocol as each member will have to completely revoke all partitions and get a whole new assignment.

However, recently, we've encountered some rebalance stuck issues, and the root cause of them are due to the out-of-date "ownedPartition". Because there are chances that the "ownedPartitions" are out-of-date, the assignors will blindly "trust" the "ownedPartitions", and do assignment depend on them, and cause unexpected results. ex: KAFKA-12984, KAFKA-13406.

Currently, we tried to workaround this issue by adding "generation" field into subscription "userData" field in cooperative sticky assignor, and deserialize them when doing assignment, to identify if the "ownedPartitions" are out-of-date or not. However, this workaround only works for cooperative sticky assignor, if users have their own custom cooperative assignor, they also need to workaround it manually. Otherwise, the same issues also happen to them. On the other hand, `StickyAssignor` is also adding "generation" field plus the "ownedPartitions" into subscription userData bytes. the difference is that the `StickyAssignor`'s user bytes also encode the prev-owned partitions while the `CooperativeStickyAssignor` relies on the prev-owned partitions on the subscription protocol directly.

Only appending "ownedPartitions" data without "generation" info in the Subscription message, is like in TCP, only send packets without appending the sequence number. It'll confuse the assignor(or TCP receivers) and make the wrong decision.

Therefore, we should add the "generation" field into "Subscription" data of the consumer protocol. (i.e. ConsumerProtocolSubscription), to allow assignor /consumer coordinator/group coordinator to have a way to identify the out-of-date members/assignments.

Public Interfaces

ConsumerProtocolSubscription

bump the version to 2, add add a new field "generationId" at the end

```
{
  "type": "data",
  "name": "ConsumerProtocolSubscription",
  // Subscription part of the Consumer Protocol.
  //
  // The current implementation assumes that future versions will not break compatibility. When
  // it encounters a newer version, it parses it using the current format. This basically means
  // that new versions cannot remove or reorder any of the existing fields.
  // Starting from version 2, we add a new field called GenerationId to indicate if the member has out-of-date ownedPartitions.    <--- new added
  "validVersions": "0-2",
  "flexibleVersions": "none",
  "fields": [
    { "name": "Topics", "type": "[]string", "versions": "0+" },
    { "name": "UserData", "type": "bytes", "versions": "0+", "nullableVersions": "0+",
      "default": "null", "zeroCopy": true },
    { "name": "OwnedPartitions", "type": "[]TopicPartition", "versions": "1+", "ignorable": true,
      "fields": [
        { "name": "Topic", "type": "string", "mapKey": true, "versions": "1+", "entityType": "topicName" },
        { "name": "Partitions", "type": "[]int32", "versions": "1+"}
      ]
    }
    { "name": "GenerationId", "type": "int32", "versions": "2+", "default": "-1"},  <-- new added
  ]
}
```

Also, there will also be a "generationId" field added into the `Subscription` class in ConsumerPartitionAssignor.

```
final class Subscription {
    private final List<String> topics;
    private final ByteBuffer userData;
    private final List<TopicPartition> ownedPartitions;
    private Optional<String> groupInstanceId;
    private final int generationId  // new added

    public Subscription(List<String> topics, ByteBuffer userData, List<TopicPartition> ownedPartitions, int
generationId) {
        this.topics = topics;
        this.userData = userData;
        this.ownedPartitions = ownedPartitions;
        this.groupInstanceId = Optional.empty();
        this.generationId = generationId;  // new added
    }

    public Subscription(List<String> topics, ByteBuffer userData, List<TopicPartition> ownedPartitions) {
        this(topics, userData, Collections.emptyList(), -1);
    }

    public Subscription(List<String> topics, ByteBuffer userData) {
        this(topics, userData, Collections.emptyList(), -1);
    }

    public Subscription(List<String> topics) {
        this(topics, null, Collections.emptyList(), -1);
    }

    // new added, the generationId getter
    public int generationId() {
        return generationId;
    }
}
```

Proposed Changes

So, during the joinGroup request, all consumers will include the "generation" data into the protocol set and send to the group coordinator. Later, when the consumer lead receive the subscription info from all consumers, it'll do the assignment based on the "ownedPartitions" and "generation" info. Also, after the assignment, we can also leverage the "ownedPartitions" and "generation" info to validate the assignments.

For built-in CooperativeStickyAssignor, if there are consumers in old bytecode and some in the new bytecode, it's totally fine, because the subscription data from old consumers will contain \[empty ownedPartitions + default generation(-1)] in V0, or \[current ownedPartitions + default generation(-1)] in V1. For V0 case, it's quite simple, because we'll just ignore the info since they are empty. For V1 case, we'll get the "ownedPartitions" data, and then decode the "generation" info in the subscription userData bytes. So that we can continue to do assignment with these information.

For built-in StickyAssignor, if there are consumers in old bytecode and some in the new bytecode, it's also fine, because the subscription data from old consumers will contain \[empty ownedPartitions + default generation(-1)] in V0, or \[current ownedPartitions + default generation(-1)] in V1. For both V0 and V1 case, we'll directly use the ownedPartition and generation info in the subscription userData bytes.

For custom assignor, they can adopt the "ownedPartitions" and "generation" info together to check if it is a stale one. If they don't update their code, it's fine since this change is backward compatible. (check below)

In the AbstractPartitionAssignor, we would have a `validateSubscription` function which takes in the ownedPartitions across all members, and needs to be called by all assignors (it is the customized assignor's own responsibility to call it), to check that ownedPartitions do not have overlaps.

Note: This KIP doesn't have anything to do with the brokers. The "generation" field added, is in the subscription metadata, which will not be deserialized by brokers. The metadata is only deserialized by consumer lead. And for the consumer lead, the only thing the lead cared about, is the highest generation of the ownedPartitions among all the consumers. With the highest generation of the ownedPartitions, the consumer lead can distribute the partitions as sticky as possible, and most importantly, without errors.

Compatibility, Deprecation, and Migration Plan

It is compatible to inject additional fields after the assignor-specific SubscriptionInfo bytes, since on serialization we would first call assignor to encode the info bytes, and then re-allocate larger buffer to append consumer-specific bytes; with the new protocol, we just need to append some existing fields before, and some new fields after the assignor-specific info bytes, and vice-versa on deserialization. So adding fields after the assignor-bytes is still naturally compatible with the plug-in assignor.

Rejected Alternatives

1. Adding a new `generation` method in `ConsumerPartitionAssignor` interface, for the assignor to get the generation info.

--> This will work for the assignor only. But actually, in `ConsumerCoordinator` , after the cooperative assignor completes its assignment, we have a validation phase, to validate if the cooperative assignor revoke partitions first before assign it to other consumers. In the validation phase, we also need the "generation" info. If the generation info only put inside assignor, the validation phase can't leverage the "generation" data.

2. in broker side, the group coordinator can check for the "generation" upon Join-Group: if it is a sentinel value (e.g. -1) then assume it is a new member that have never been in the group yet, and hence always set the current generation; if it is not sentinel value and stale, then return the ILLEGAL_GENERATION error directly. And, upon getting such error the member should not clear its member ID if there's one, but only reset the generation to null and also clean up the ownedPartitions before re-joining.

--> reject this suggestion because this way, the member needs 1 more round to join group. That is:

1. member-A joined group with generation 1 metadata (Subscription Info)

2. broker checked the generation ID, and found it's out-of-date (current generation id is 3), return `ILLEGAL_GENERATION`

3. member-A got the `ILLEGAL_GENERATION` error, clear the old ownedPartition and rejoin again

4. continue the rebalance

We can see, the purpose of step 1 ~ 3 is to clear the old metadata (i.e. ownedPartition) in the consumer side. However, we can identify the out-of-date ownedPartition via the SubscriptionInfo when doing assignment. This way, the out-of-date member doesn't need to rejoin group, and reduce the network traffic. ex:

1. member-A joined group with generation 1 metadata (Subscription Info) (current generation id is 3)

2. continue the rebalance

3. consumer lead is performing assignment, parsing the subscription info, and ignore (or do other handling) the old generation ownedPartitions, and continue the assignment.

4. complete the assignment

This can still complete the assignment without any error, and reduce the network traffic between consumer/broker.

3. One scenario that we need to consider is what happens during a rolling upgrade. If the coordinator moves back and forth between brokers with different IBPs, it seems that the same epoch numbers could be reused for a group, if things are done in the obvious manner (old IBP = don't read or write epoch, new IBP = do).

--> This KIP doesn't care about the group epoch number at all. The subscription metadata is passed from each members to group coordinator, and then pass all of them back to consumer lead. So even if the epoch number is reused in a group, it should be fine. On the other hand, the group coordinator will have no idea if the join group request sent from consumer containing the new subscription "generation" field or not, because group coordinator won't deserialize the metadata.