KIP-814: Static membership protocol should let the leader skip assignment

Status

Current state: *"Accepted"*

Discussion thread: *here*

JIRA: *here*

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

Motivation

Apache Kafka 2.4.0 has introduced the static membership protocol to reduce consumer rebalances (KIP-345). The protocol allows a member to briefly leave and to rejoin the group without triggering and/or requiring a rebalance. The current implementation has a limitation though. When the leader of the group leaves and re-joins the group, the group coordinator updates its state to make it the new leader of the group but it does not inform the client about it. We trick the protocol by not telling the new leader that it is the leader in order to avoid computing a new assignment that would not be propagated to the other group members. The group end up with no active leader on the client side. This is problematic because the leader is responsible for monitoring the metadata changes and for triggering a rebalance if they change. Without an active leader, the group won't catch any metadata changes (e.g. new partitions).

Proposed Changes

We basically want to re-joining leader to know that it is the current leader of the group while avoiding computing a new assignment. We propose to achieve this by adding a `SkipAssignment` field in the `JoinGroupResponse`. When the group coordinator detects that the static leader is rejoining the group, it will set `SkipAssignment` to true, set the correct leader id and provide subscriptions of all members (which might be different from the leader's own subscriptions) so that the leader can monitor all topics subscribed in the group. Then the leader will send an `SyncGroup` with no assignment to collect its own (and existing) assignment. The group coordinator will do so only for consumers that support the latest version of the `JoinGroup` API. Other consumers will continue to behave as they do today.

Note that the proposed solution is not perfect. The remaining issue is that the leader can not detect metadata changes (e.g. new partition added) while it is offline. Adding partitions to topics is a rather infrequent operations so the likelihood for it to happen exactly when the static leader is down is low. The proposed solution seems to be acceptable as a first step to fix the common issue.

Public Interfaces

The `SkipAssignment` is added to the `JoinGroupResponse`.

JoinGroupResponse

```
{
  "apiKey": 11,
  "type": "response",
  "name": "JoinGroupResponse",
  // Version 1 is the same as version 0.
  //
  // Version 2 adds throttle time.
  //
  // Starting in version 3, on quota violation, brokers send out responses before throttling.
  //
  // Starting in version 4, the client needs to issue a second request to join group
  // with assigned id.
  //
  // Version 5 is bumped to apply group.instance.id to identify member across restarts.
  //
  // Version 6 is the first flexible version.
  //
  // Starting from version 7, the broker sends back the Protocol Type to the client (KIP-559).
  //
  // Version 8 adds the SkipAssignment field.
  "validVersions": "0-8",
  "flexibleVersions": "6+",
  "fields": [
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "2+", "ignorable": true,
      "about": "The duration in milliseconds for which the request was throttled due to a quota violation, or
zero if the request did not violate any quota." },
    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The error code, or 0 if there was no error." },
    { "name": "GenerationId", "type": "int32", "versions": "0+", "default": "-1",
      "about": "The generation ID of the group." },
    { "name": "ProtocolType", "type": "string", "versions": "7+",
      "nullableVersions": "7+", "default": "null", "ignorable": true,
      "about": "The group protocol name." },
    { "name": "ProtocolName", "type": "string", "versions": "0+", "nullableVersions": "7+",
      "about": "The group protocol selected by the coordinator." },
    { "name": "Leader", "type": "string", "versions": "0+",
      "about": "The leader of the group." },
    // New Field //
    { "name": "SkipAssignment", "type": "bool", "versions": "8+", "default": "false",
      "about": "True is the leader must skip running the assignment." },
    { "name": "MemberId", "type": "string", "versions": "0+",
      "about": "The member ID assigned by the group coordinator." },
    { "name": "Members", "type": "[]JoinGroupResponseMember", "versions": "0+", "fields": [
      { "name": "MemberId", "type": "string", "versions": "0+",
        "about": "The group member ID." },
      { "name": "GroupInstanceId", "type": "string", "versions": "5+",
        "nullableVersions": "5+", "default": "null",
        "about": "The unique identifier of the consumer instance provided by end user." },
      { "name": "Metadata", "type": "bytes", "versions": "0+",
        "about": "The group member metadata." }
    ]}
  ]
}
```

Compatibility, Deprecation, and Migration Plan

The change is backward compatible as it is only applied to new consumers (with bumped API). Old consumers continue to behave as they do today.

Rejected Alternatives

1. The group coordinator could trigger a rebalance when the static leader re-joins the group. The major advantage is that it would work for all client versions. The downside is that it would trigger a rebalance every time the leader leaves and re-joins the group. That goes against the initial goal of the static membership protocol.

2. The group coordinator could parse all the assignments in the group, listen to metadata changes for the partitions, and trigger a rebalance when required. That would be clean. The main limitation is that the group coordinator does not parse the assignment as of today. They are just a bunch of bytes from its point of view. Changing this is a significant change in the protocol philosophy and a significant amount of work.
3. When the leader re-joins the group, the group coordinator could tell the leader that it is new leader of the group (like we do for all other cases), give it all the subscriptions, and let it compute a new assignment. That would work if the assignors are deterministic. Unfortunately, assignors are most of the time deterministic given the exact same input. The issue is that the input of the assignor is an `HashMap` in the Java client so the ordering is not guarantee at all, especially because the re-joining leader has a new member id. Moreover, the order in the `JoinGroupResponse` is not guaranteed neither.
4. Subscription changes are monitored by all the members whereas metadata changes are only monitored by the leader. We could relax this constrain and have all members monitor metadata changes as well. The main drawback of this option is that it could be chatty on the wire because all members would try to trigger a rebalance when the metadata changes.