Consumer rebalance

Summary

One of the main reasons we are refactoring the KafkaConsumer is to satisfy the requirements of the new rebalance protocol introduced in KIP-848.

KIP-848 contains two assignment modes, server-side mode and client-side mode. Both use the new Heartbeat API, the *ConsumerGroupHeartbeat*.

The server-side mode is simpler: the assignments are computed by the Group Coordinator, and the clients are only responsible for revoking and assigning the partitions.

If the user chooses to use the client-side assignor, the assignment will be computed by one of the member, and the assignment and revocation is done via the heartbeat as server side mode.

In the new design we will build the following components:

1. **GroupState**: keep track of the current state of the group, such as *Generation*, and the rebalance state.
2. **HeartbeatRequestManager**: A type of request manager that is responsible for calling the ConsumerGroupHeartbeat API
3. **Assignment Manager:** Manages partition assignments.

Rebalance Flow

New Consumer Group

1. The user invokes subscribe(). SubscriptionState is altered. A subscription state alters event is sent to the background thread.
2. The background thread processes the event and updates the GroupState to PREPARE.
3. HeartbeatRequestManager is polled. It checks the GroupState and determines it is time to send the heartbeat.
4. ConsumerGroupHeartbeatResponse received. Updated the GroupState to ASSIGN.
5. PartitionAssignmentManager is polled, and realize the GroupState is in ASSIGN. Trigger assignment computation:
6. [We might need another state here]
7. Once the assignment is computed, send an event to the client thread to invoke the rebalance callback.
8. Callback triggered; notify the background thread.
9. PartitionAssignmentManager is polled Transition to Complete.
10. [something needs to happen here]
11. Transition the GroupState to Stable.

GroupState

[UNJOINED, PREPARE, ASSIGN, COMPLETE, STABLE]

- UNJOINED: There's no rebalance. For the simple consumed use case, the GroupState remains in UNJOINED
- PREPARE: Sending the heartbeat and await the response
- ASSIGN: Assignment updated, client thread side callbacks are triggered, and await completion
- COMPLETE: Client thread callback completed and has notified the background thread.
- STABLE: stable group

Consumer group member state machine

It becomes clear when reading KIP-848 that the work of keeping the consumer group in proper state is fairly involved. We therefore turn our focus now to the logic needed for the *consumer group member state machine* (hereafter, CGMSM).

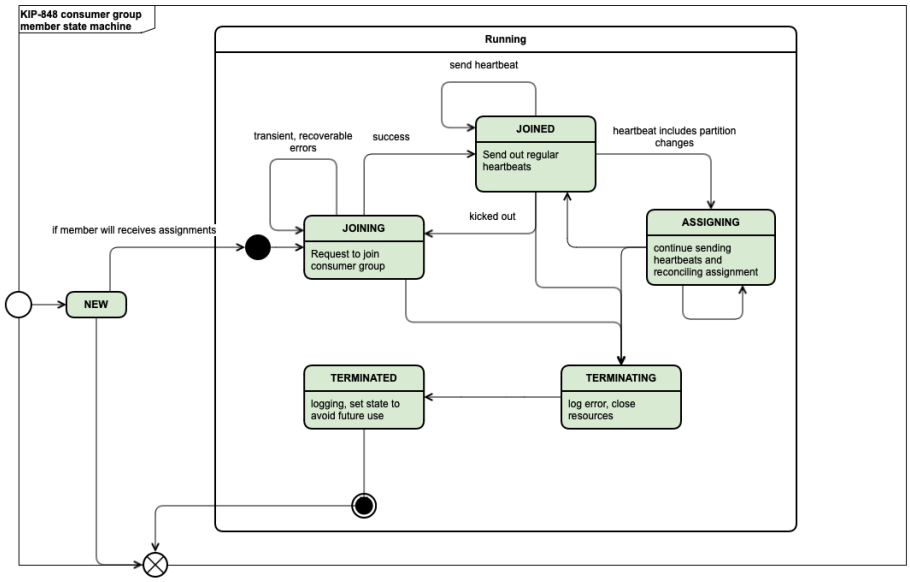
Based on the user calling either `assign()` or `subscribe()`, a `Consumer` determines how topic partitions are to be assigned. If the user calls the `subscribe()` API, the `Consumer` knows that it is being directed to use Kafka's *consumer group*-based partition assignment. The use of `assign()` signifies the user's intention to manage the partition assignments from within the application via *manual* partition assignment. It is only in the former case that a CGMSM needs to be created.

Note that the necessary logic to establish a connection to the Kafka broker node acting as the group coordinator is outside the scope of the CGMSM logic.

In order to keep the size of a `ConsumerGroupHeartbeatRequest` smaller, KIP-848's description of the request schema states that some values are conditionally sent with the request only when they change on the client. These values include:

- `InstanceId`
- `RackId`
- `RebalanceTimeoutMs`
- `SubscribedTopicNames`
- `SubscribedTopicRegex`
- `ServerAssignor`
- `ClientAssignors`
- `TopicPartitions`

The following diagram provides a visual overview of the states and transitions for members of the consumer group:



The following description provides more clarity on the states that make up the CGMSM:

NEW

`NEW` is the initial state for a CGMSM upon its creation. The `Consumer` will remain in this state until the next pass of the background thread loop.

JOINING

A state of `JOINING` signifies that a Consumer wants to join a consumer group. On the next pass of the background thread, the `Consumer` will enter this state to begin communicating with the Kafka broker node that was elected as the group coordinator. A `ConsumerGroupHeartbeatRequest` will be sent to the coordinator with specific values in the request:

- `MemberId` is set to `null`
- `MemberEpoch` is set to the hard-coded value of `0`

Since this is the first request to the coordinator, the CGMSM will include a `ConsumerGroupHeartbeatRequest` with all conditional values present. This includes setting `TopicPartitions` to `null` since there are no assigned partitions in this state.

Once the initial `ConsumerGroupHeartbeatResponse` is received successfully, the CGMSM will update its local `MemberId` and `MemberEpoch` based on the returned data. It will then transition to the `JOINED` state.

JOINED

The `JOINED` state simply indicates that the `Consumer` instance is known to the coordinator as a member of the group. It does not necessarily imply that it has been assigned any partitions. While in the `JOINED` state the CGMSM will periodically send requests to the coordinator at the needed cadence in order to maintain membership.

The CGMSM should transition back to the `JOINING` state if the `ConsumerGroupHeartbeatResponse` has an error of `UNKNOWN_MEMBER_ID` or `FENCED_MEMBER_EPOCH`. If either of those errors occur, the CGMSM will clear its "assigned" partition set (without any revocation), and transition to the `JOINING` set so that it rejoins the group with the same `MemberId` and the `MemberEpoch` of 0.

The CGMSM will transition into the `ASSIGNING` state when the `ConsumerGroupHeartbeatResponse` contains a non-`null` value for `Assignment`.

ASSIGNING

The `ASSIGNING` state is entered with the intention that the CGMSM will need to perform the assignment reconciliation process. As is done in the `JOINED` state, the CGMSM will continue to communicate with the coordinator via the heartbeat mechanism to maintain its membership.

The first action that is performed in this state is to update the CGMSM's value for the member epoch as provided in the `ConsumerGroupHeartbeatResponse`.

Next, the CGMSM performs a comparison between its *current* the assignment and the value of `Assignment` contained in the `ConsumerGroupHeartbeatResponse`. If the two assignments are equal, the CGMSM has reconciled the assignment successful and will transition back to the `JOINED` state. If they are not equal, the reconciliation process begins.

KIP-848 states that during reconciliation, partitions are revoked first and then assigned second, as two distinct steps.

Partition revocation involves:

1. Removing the partitions from the CGMSM's "assigned" set
2. Commits the offsets for the revoked partitions
3. Invokes ConsumerRebalanceListener.onPartitionsRevoked()

Partition assignment includes:

1. Adding the partitions to the CGMSM's "assigned" set
2. Invokes ConsumerPartitionAssignor.onAssignment(), if one is set
3. Invokes ConsumerRebalanceListener.onPartitionsAssigned()

Questions

1. Do we need to heartbeat between revocation and assignment? YES, I think so.
2. Do we want to split up `ASSIGNING` into separate states `REVOKING` and `ASSIGNING`?

`TERMINATING`

TBD

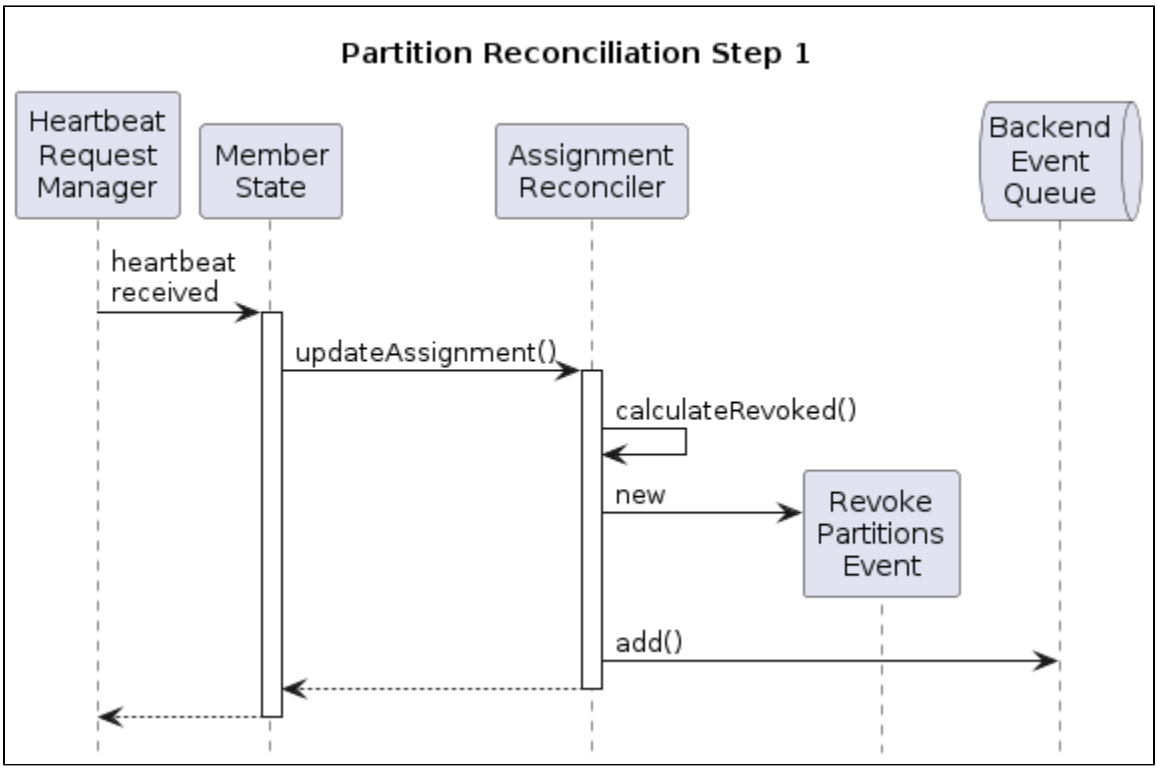
`TERMINATED`

TBD

Partition Reconciliation

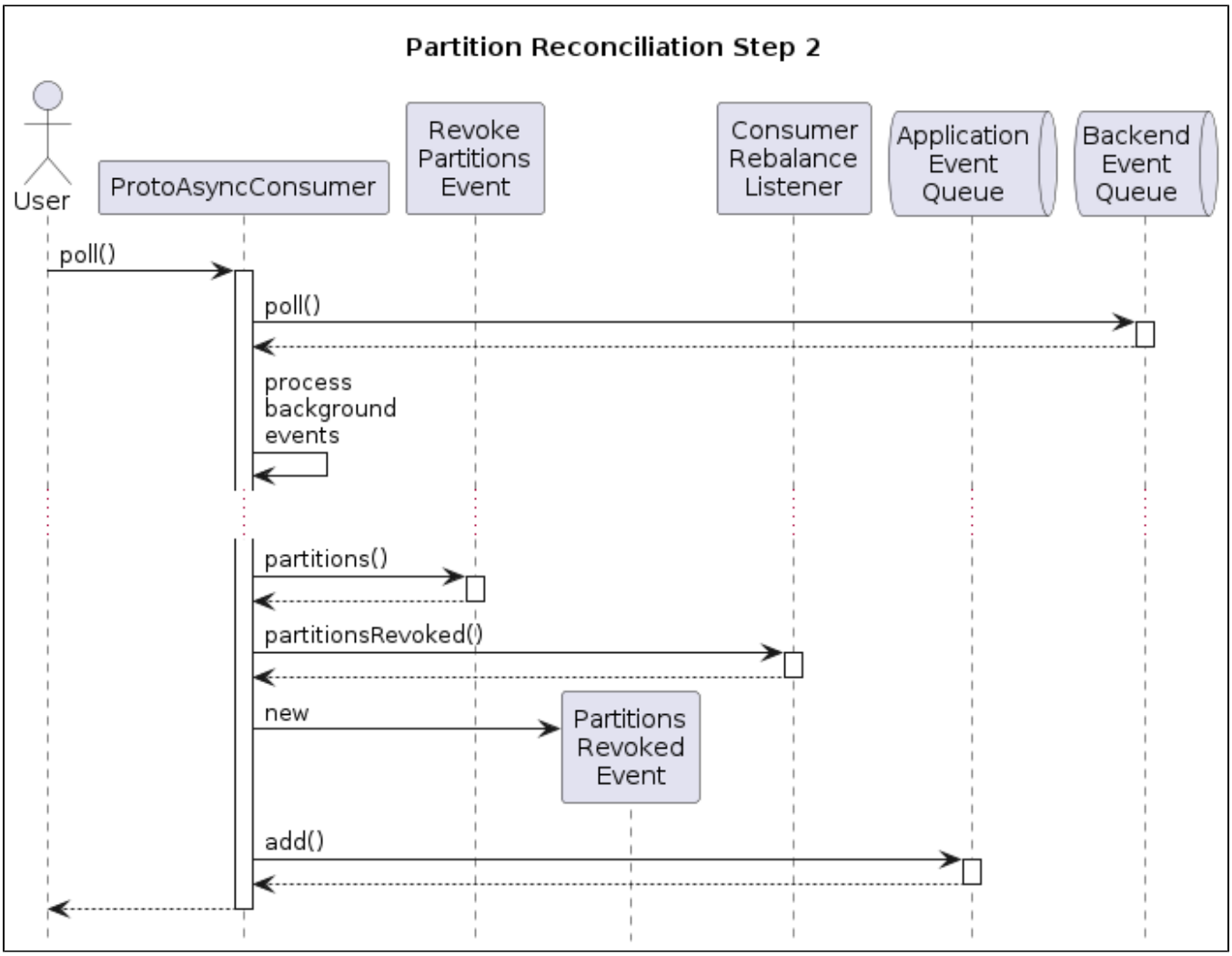
Partition reconciliation is the act of updating the consumer's internal state to reflect its assigned partitions. This reconciliation occurs in multiple steps, shown here:

Step 1

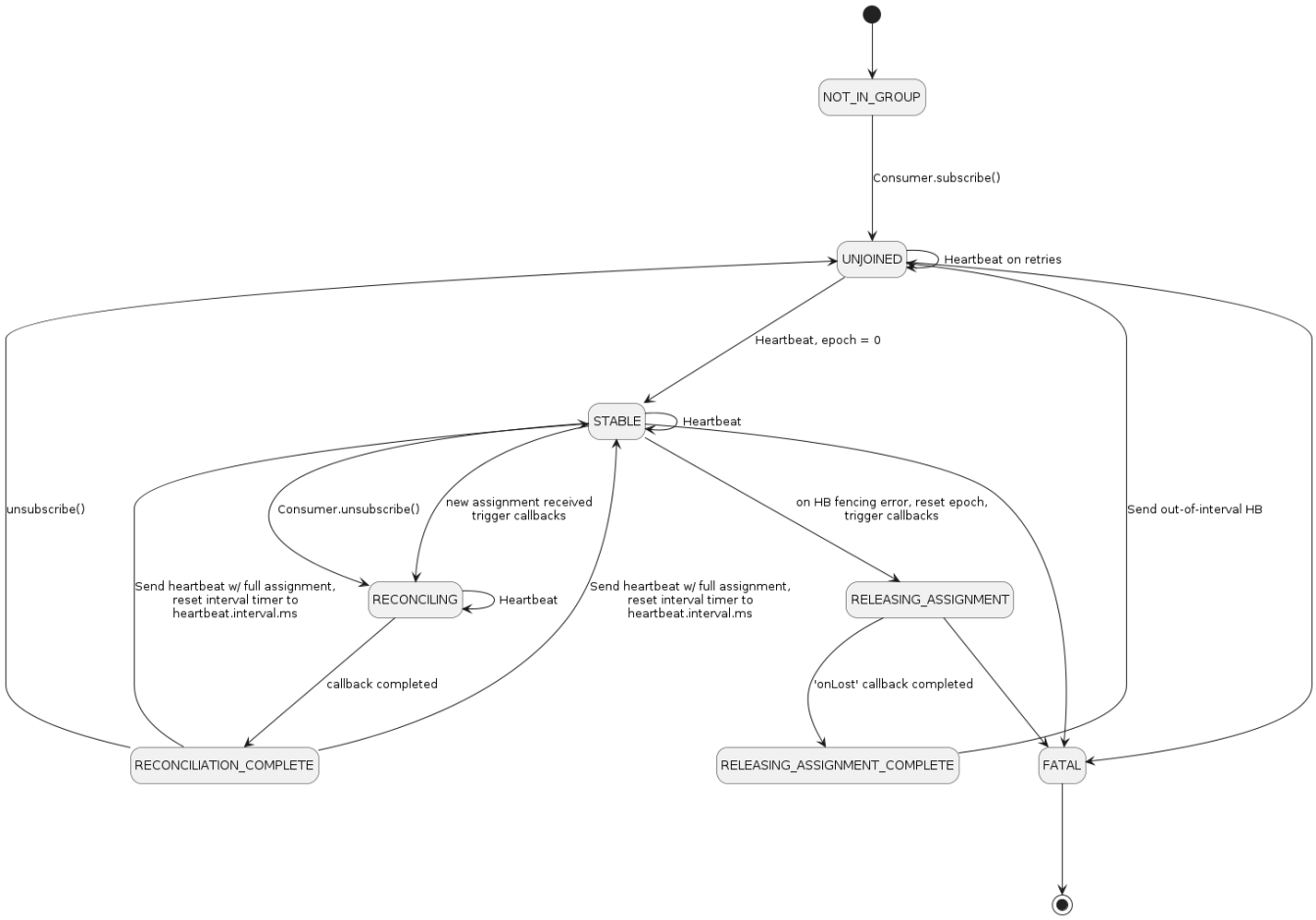


In the above...

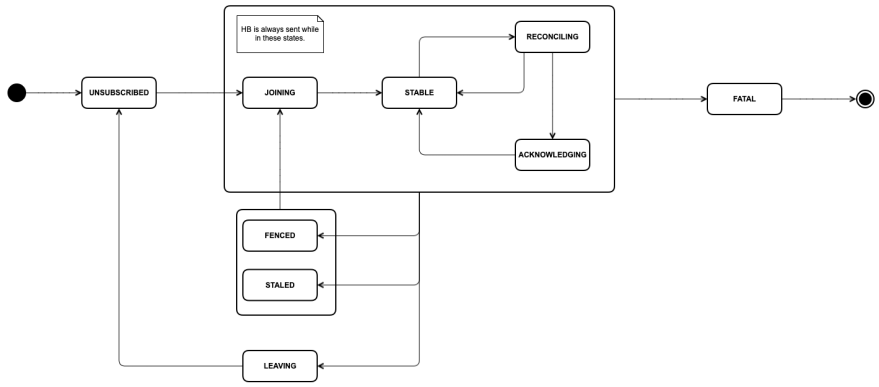
Step 2



Rebalance State Machine



David's notes



States:

- UNSUBSCRIBE: The consumer is not subscribed to any topics nor regex therefore it is not part of a consumer group.
- JOINING: The consumer has subscribed with topic names or a regex. Consumer send an HB request to join the group with epoch 0 and transitions to Stable.
- STABLE: While in this state, has nothing to do besides heartbeatting to remain in the group.
- RECONCILE_ASSIGNMENT: Whenever the consumer received a non-null assignment from the group coordinator, it transitions to this state and reconciles its assignment. It should revoke unnecessary partitions and assign the new ones. This also commits offsets and triggers the rebalance callbacks. When the reconciliation completes, it transitions to ACK_ASSIGNMENT.
- ACK_ASSIGNMENT: This signals to the HB manager than an HB request must be sent in the next run of the event loop event the HB internal has not expired. It transitions to STABLE when that signal is given.

- UNSUBSCRIBING: When the consumer calls unsubscribe or close (this can happen anytime), it transitions to this state, cancels any ongoing reconciliation (how to?), revoke partitions/commit offsets and send the last HB to leave the group. When done, it transitions to UNSUBSCRIBE.
- FENCED: When the group coordinator fences the member (this can happen anytime), it transition to this state, cancels any ongoing reconciliation (how to?), resets the member epoch and invokes onLost for all partitions. When done, it transitions to JOINING to rejoin the group.
- FATAL: The consumer enters this state whenever a fatal errors is encountered. This is not recoverable.

Notes

- When the subscriptions are changed, should we send the next HB immediately?
- Should we transition from FATAL to UNSUBSCRIBE when the subscriptions are changed? Let's imagine that the user subscribes with an invalid regex. In this case, the consumer transition to FATAL as this is not recoverable. However, the user may react to the exception and change the subscriptions. We may need to give it another try if we have new subscriptions.