KIP-822: Optimize the semantics of KafkaConsumer#pause to be consistent between the two RebalanceProtocols

Status

Current state: *Under Discussion*

Discussion thread: here

JIRA: Unable to render Jira issues macro, execution error.

Motivation

When users use the kafkaConsumer#pause(...) method, they will maybe ignore: the pause method may no longer work, and data will be lost.
In short, during the rebalance of the group, ConsumerCoordinator#invokePartitionsRevoked(...) will clear the paused mark on the partitions previously held by kafkaConsumer. However, while clearing the paused mark of partitions, the corresponding message in the memory (Fetcher.completedFetches) of pausedPartitions was not cleared, resulting in Fetcher#fetchedRecords() still fetching the message and returning it to the customer.
In fact, for consumers who use the RebalanceProtocol.COOPERATIVE protocol
For example, consumers who use the currently supported PartitionAssignor: CooperativeStickyAssignor, through code analysis, we can find that the default behavior of these consumers is to maintain the old paused flag, and consumers who use the RebalanceProtocol.EAGER protocol default to clear all paused marks.
I suggest that the KafkaConsumer behavior of the two RebalanceProtocol should be consistent, otherwise it will cause ambiguity to the existing KafkaConsumer#pause(...) and cause great confusion to users.

Public Interfaces

<1>In the ConsumerCoordinator#onJoinPrepare(...) method, record all pausedTopicPartitions from the current assignment of KafkaConsumer;

<2> In the ConsumerCoordinator#onJoinComplete(...) method, use pausedTopicPartitions to render the latest assignment and restore the paused marks of the partitions that are still in the latest assignment.

Note: If the new assignment of kafkaConsumer no longer contains topicPartitions that have been paused before rebalance, the paused mark of these topicPartitions will be lost forever on the kafkaConsumer side, even if in a future rebalance, the kafkaConsumer will hold these partitions again.

Proposed Changes

1) When rebalance starts to prepare, add new logic to ConsumerCoordinator#onJoinPrepare(...)

Before executing invokePartitionsRevoked(...) and subscriptions.assignFromSubscribed(...), filter out customerPausedPartitions from the subscriptions. assignment of the current KafkaConsumer, and customerPausedPartitions should be instance variables of ConsumerCoordinator.

```
customerPausedPartitions = subscriptions.pausedPartitions();
//Add new code in front of the following two codes

exception = invokePartitionsRevoked(...);
subscriptions.assignFromSubscribed(...);
```

2) After the rebalance is completed, add new logic to ConsumerCoordinator#onJoinComplete(...)

```
        subscriptions.assignFromSubscribed(assignedPartitions);

        //Add new code here
        if (customerPausedPartitions != null && customerPausedPartitions.size() != 0){
            customerPausedPartitions.forEach(topicPartition -> {
                if(subscriptions.isAssigned(topicPartition))
                    subscriptions.pause(topicPartition);
            });
            customerPausedPartitions = null;
        }

        // Add partitions that were not previously owned but are now assigned
        firstException.compareAndSet(null, invokePartitionsAssigned(addedPartitions));
```

Compatibility, Deprecation, and Migration Plan

This is an optimization of the semantics of the kafkaConsumer pause method, so that it can behave consistently under different RebalanceProtocols

Rejected Alternatives

Here are a few other suggestions
suggestion 1: Code changes may be more complex and also introduce additional network traffic
suggestion 2-5: Aim to provide a KafkaConsumer#Pause method that is not affected by groupRebalance, and the changes are also larger, which will greatly change the existing behavior

1ConsumerCoordinator#invokePartitionsRevoked should also trigger Fetcher to clean up the revokedAndPausedPartitions message in memory when clearing the paused mark

This can prevent the Fetcher#fetchedRecords() method from mistakenly thinking that revokedAndPausedPartitions is legal and returning messages. There are various checks on the partition in the fetchedRecords method.

The price of this is that if the user does not call the pause(...) method before calling the poll method next time, a new FetchMessage request may be initiated, which will cause additional network transmission.

2In the groupRebalance process, pass the paused flag of topicPartitions

In the JoinGroup request, in addition to reporting the topic that it wants to subscribe to, each consumerMember should also report its pausedTopicPartitions. The JoinGroup response received by the LeaderConsumer should contain all paused partitions under the entire group.

The latest assignment made by LeaderConsumer should maintain the paused mark and be packaged in LeaderConsumer's SyncGroup request

In this way, after groupRebalance is completed, even if a paused topicpartition is assigned to a new consumer, the new consumer can continue to maintain the paused mark.

The KafkaConsumer#paused() method can return the partitions that KafkaConsumer did not call the pause(Collection<TopicPartition> partitions) method.

3KafkaConsumer provides a pause method for topic level and supports regular expressions

KafkaConsumer#pause(Collection<String> topics)

KafkaConsumer#pause(Pattern pattern)

Similar to the paused mark in SubscriptionState.assignment, we need to provide a new instance variable ‘TopicState’ in SubscriptionState to store the topic-level paused mark. The ‘TopicState’ data structure can refer to the existing TopicPartitionState.

<1> ‘TopicState’ should not be affected by groupRebalance, and the paused mark in TopicState will not be changed during the groupRebalance process. TopicState should be the memory mark of a single KafkaConsumer, and it does not have to be passed to other consumers after the rebalance is completed.

<2> pause(Collection<String> topics), throws IllegalStateException if this consumer is not currently subscribed to any topic provided

<3> Fetcher's fetchedRecords() and sendFetches() can be combined with TopicState considerations to decide whether to return a message to the user or initiate a Fetch request

<4> Provide KafkaConsumer#resume(Collection<String> topics) and KafkaConsumer#resume(Pattern pattern) methods to clean up topic-level paused marks.

4KafkaConsumer provides a pause method for the consumer level

KafkaConsumer#pause()

The existing pause method is for topicPartition and may sometimes be too fine-grained. And the paused mark is bound in the assignment, it is inevitable that it will not be affected by groupRebalance.

<1> This method may also be the user's most urgent need. After calling this pause() method, kafkaConsumer will mark itself as a paused state, and the poll method will determine the value of isKafkaConsumerPaused to decide whether to return a message to the user or initiate a Fetch request. This isKafkaConsumerPaused mark should also be held by a single KafkaConsumer itself.

<2> Users do not need to worry about the poll method returning data after calling the KafkaConsumer#pause() method.

Users can always call the poll method to avoid the following two results if kafkaConsumer does not call the poll method for a long time

<3> Provide KafkaConsumer#resume() at the kafkaConsumer level to clean up the paused mark of KafkaConsumer

5 Strip the paused mark of topicpartition from assignment

<1> The new instance variable TopicPartitionPausedState is used in SubscriptionState to store the paused mark of each topicPartition, and the paused mark is not stored in the assignment.

<2> In my opinion, the pause and resume methods are entirely the behavior of the kafkConsumer client, and it should not be affected by groupRebalance. During the groupRebalance process, KafkaConsumer will not silently modify TopicPartitionPausedState. TopicPartitionPausedState can only be modified by the user's pause and resume.

At the same time, it supports the user to pause a topicPartition that is not in the assignment, because the paused mark is only the concept of the partition setting of the kafkaConsumer.

In other words, no matter whether the consumer has assigned any topicPartitions or not, kafkaConsumer can pause any topicPartition, even if the topicPartition has not been existed, it may be created(or by addPartition) in the future.

<3> The resume(Collection<TopicPartition> partitions) method, clean up the paused mark in TopicPartitionPausedState