KIP-275 - Indicate "isClosing" in the SinkTaskContext

Status

Current state: Discarded

Discussion thread: here

Vote thread: here

JIRA: KAFKA-6725

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

Motivation

We've been working on deploying Kafka Connect with the S3 connector as an archiver for Avro records coming over some topics in our Kafka clusters. Because of the way we want to use the archives later, we've configured our tooling to put up to 100,000 records in a single Avro file in the S3 bucket. While we were working on this, we experienced some connectivity issues to S3. Every time one of these connectivity failures occurred, the S3 client library would eventually throw a TimeoutException, the Connector task would die, and a rebalance would be triggered.

That rebalance would cause all *other* tasks to lose any forward progress they had made since their last commit and rewind. Because of the nature of our intermittent connection issues we would actually see this happen frequently enough to end up lagged by hundreds of millions of records.

To put it another way what we were seeing was something like:

1. Connect spins up with Tasks 1 thru 4
2. Task 1, 2, 3 each consume about 50,000 records. Task 4 consumes 100,000.
3. Task 4 decides to try and commit to S3, but starts timing out.
4. In the interim Task 1, 2, 3 each consume 10,000 more records.
5. Task 4 finally throws a TimeoutException from the S3 client and crashes
6. A rebalance triggers. Task 1, 2, and 3 are told to close and get no signal that they should attempt to commit what they have. They drop the records they have consumed.
7. The rebalance finishes and Task 1, 2, and 3 come back up in the same state they were in when the connector first spun up. At this point, they have to re-consume the records they already tried to consume.

Due to the connectivity issues this process repeated many, many times. Commits from some nodes would work fine, others wouldn't. But the nodes that were not having connectivity issues would still lose forward progress when the rebalance was triggered.

After some code analysis, we determined that there wasn't an existing API that a Connector could use to determine that the task was about to be shut down for a rebalance. After some thought we decided that this was a problem more generally about giving connectors the information about the state of the world outside the connector such that they could make intelligent decisions about how to behave in `preCommit`. Giving this information to a connector gives developers writing connectors options to change their behavior if a rebalance or shutdown is pending.

To achieve this we would like to add such an API so that the `preCommit` hook of a Connector can decide if it should behave differently because the task is actively trying to shut down.

Public Interfaces

To accomplish this goal we're proposing the addition of:

- A single method, `isClosing`, to the `SinkTaskContext` interface that will return a boolean
  - This method will return `true` if preCommit is being invoked as a part of a rebalance or shutdown and the task is about to be closed.
  - It will be `false` at all other times.

Proposed Changes

1. We'll add the `isClosing` method to the `SinkTaskContext` as mentioned above.
2. We'll add the implementation for that to `WorkerSinkTaskContext`
3. We'll add a setter named `setClosingInProgress` to `WorkerSinkTaskContext` that'll change the internal state
4. In `WorkerSinkTask.commitOffsets`:

    a. If the closing parameter is set to `true`, we'll invoke `setClosingInProgress` on the context to `true`. This will occur before the invocation of `preCommit`.
    b. Before `commitOffsets` returns, it will `setClosingInProgress` back to `false`.
5. Add a note to the `preCommit` javadoc to indicate the importance of checking `SinkTaskContext.isClosed` when running,

This change will give preCommit the opportunity to provide some additional commits that it would like Kafka Connect to make before the task is fully shut down.

Compatibility, Deprecation, and Migration Plan

This is simply the addition of an additional method on an existing interface. Sinks that wish to use it can invoke it. Sinks that do not, don't have to do anything. As a result there are no migration or compatibility concerns.

Rejected Alternatives

- We considered adding a closing parameter to preCommit, but that would be a breaking change unless it was done very carefully. The additional complexity would buy us marginal extra value, if any at all, so we decided to go with the addition to the context object.