KIP-572: Improve timeouts and retries in Kafka Streams

Status

Current state: *"Accepted" [VOTE] KIP-572: Improve timeouts and retires in Kafka Streams*

Discussion thread: [DISCUSS] KIP-572: Improve timeouts and retires in Kafka Streams

JIRA: **KAFKA-9274** - Getting issue details... STATUS

Released: 2.7 / 2.8

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

Motivation

Today, Kafka Streams relies mainly on its internal clients (consumer/producer/admin) to handle timeout exceptions and retries (the "global thread" is the only exception). However, this approach has many disadvantages. (1) It is harder for users to configure and reason about the behavior and (2) if a client retries internally, all other tasks of the same `StreamThread` are blocked. Furthermore, the Kafka Streams `retries` config has a default value of 0 and is only used in the global thread while producer and admin client default `retires` is `Integer.MAX_VALUE` (note that the embedded clients in Kafka Streams also use `MAX_VALUE` as default; the default value of `retries=0` only applies to the global thread). This implies that if users set Kafka Streams `retries` they may accidentally reduce the producer and admin client retry config.

To make Kafka Streams more robust, we propose to catch all client `TimeoutExceptions` in Kafka Streams and handle them more gracefully. Furthermore, reasoning about time is simpler for users then reasoning about number of retries. Hence, we propose to base all configs on timeouts and to deprecate `retries` configuration parameter for Kafka Streams**.**

Public Interfaces

We propose to deprecate the `retries` configuration parameter for **Kafka Streams**. Furthermore, we introduce `task.timeout.ms` as an upper bound for any task to make progress with a default config of 5 minutes. If a task hits a client TimeoutException, the task would be skipped and the next task is processed.

The existing `retry.backoff.ms` is used as backoff time (default value 100ms) if a tight retry loop is required. We rely on client internal retry/backoff mechanism to void busy waiting (cf. KIP-580: Exponential Backoff for Kafka Clients).

Proposed Changes

Kafka Streams will ignore the `retires` config and we only keep it to not break code that might set it and log a warning if used. The default `retries` value in Kafka Streams is 0 and we want to have a more robust default configuration. Note that the default `retries` values of 0 does not apply the embedded producer or admin client. Only if the user explicitly sets `retries` the embedded producer and admin client configs would we changed (this KIP does not change this behavior).

Furthermore, we propose to catch all client `TimeoutException` in Kafka Streams instead of treating them as fatal, and thus to not rely on the consumer/producer/admin client to handle all such errors. If a `TimeoutException` occurs, we skip the current task and move to the next task for processing (we will also log a WARNING for this case to give people inside which client call did produce the timeout exception). The failed task would automatically be retired in the next processing loop. Because other tasks are processed until a task is retried, we don't have to worry about a busy wait situation. Even if a thread would have only a single task, the clients internal exponential retries would avoid busy waiting.

To make sure that timeout issues can be reported eventually, we use a new `task.timeout.ms` config to allow user to stop processing at some point if a single task cannot make any progress. The "timer" for `task.timeout.ms` starts when the first client `TimeoutException` is detected and is reset/disabled if a task processes records successfully in a retry. If `task.timeout.ms` passed, a final attempt will be made to make progress (this strategy ensures that a task will be retried at least once; except `task.timeout.ms` is set to 0, what implies zero retries); if another client `TimeoutException` occurs, processing is stopped by re-throwing it and the streams-thread dies. Note that the `task.timeout.ms` config does only apply if a `TimeoutException` occurred. During normal, potentially slow processing, `task.timeout.ms` would not be applied.

Note that some client calls are issued for multiple tasks at once (as it is more efficient to issue fewer requests to brokers). For this case, the "timer" would start ticking for all those tasks.

To replace `retries` in the global thread's initialization phase, we also retry TimeoutException until `task.timeout.ms` expires. We apply existing `retry.backoff.ms` config and rely on the client to do exponential backoff and retry for this case.

Last, the admin client is used within the group leader to collect topic metadata and to create internal topics if necessary. If those calls fails, they are retried within Kafka Streams re-using the admin client's `retries` config. The current retry loop is across multiple admin client calls that are issues interleaved. This interleaved retry logic should be preserved. However, we should not retry infinitely (and also not allow users to specify how long to retry) to avoid that the leader is stuck forever (even if it would be removed from the group by the group coordinator after a timeout anyway that is set to `max.poll.interval.ms`). To avoid dropping out of the consumer group, the retry loop should be stopped before we hit the timeout. We propose to use a 50% threshold, i.e., half of `max.poll.interval.ms`.

Compatibility, Deprecation, and Migration Plan

Kafka Streams will ignore `retries` config; however, the new default will be more robust and thus no backward compatibly concern arises. If users really want to have the old "non robust" fail immediately behavior, they can set `task.timeout.ms=0`.

Test Plan

Regular unit and integration tests are sufficient. Existing system tests should provide good coverage implicitly.

Rejected Alternatives

- Reuse the existing `retries` config and handle client TimeoutException based on it. Rejected because a reasoning about time is easier for users and other client started to move away from count based retries already.
- A task could be retried immediately if a client TimeoutException occurs instead of skipping it. However, this would result is "busy wait" pattern and other tasks could not make progress until the "failing" task makes progress again of eventually times out.
- It would be possible to apply `retries` on a per method level (ie, for each client method that is called, an individual retry counter is maintained). This proposal is rejected because it seems to be too fine grained and hard to reason about for users.

- If would be possible to apply `retries` at the thread level, i.e., whenever the thread does not make any progress in one task-processing-loop (ie, all tasks throw a timeout exception within the loop), the per-thread retry counter would be increased. This proposal is rejected as too coarse grained. In particular, a single task could get stuck while other tasks make progress and this case would not be detected.
- If would be possible to apply `thread.timeout.ms` at the thread level instead of a `task.timeout.ms` at a task level: whenever the thread does not make any progress on any tasks within the timeout, the thread would fail. This proposal is rejected as too coarse grained. In particular, a single task could get stuck while other tasks make progress and this case would not be detected.
- To distinguish between retries within Kafka Streams and client retries (in particular the producer's send `retries` config), we could add a new config (eg, `task.retries`). However, keeping the number of config small is desirable and the gain of the new config seems limited.
- To avoid that people need to consider setting `producer.retries` and `admin.retires` explicitly, we could change the behavior of Kafka Streams and use `retries` explicitly for Streams level retries. For this case, setting `retries` would not affect the producer or admin client and both retries could only be change with their corresponding client-prefix config. This would be a backward incompatible change.
- We considered to deprecate the `retries` configuration parameter also for the producer and admin client. However, there are some use cases that need to disable retries all together what is no possible by setting producer/admin client timeouts to zero (in contrast to the new `task.timeout.ms=0` setting that disables retrying).