KIP-288: [DISCARDED] Consumer.poll() timeout semantic change and new waitForAssignment method

Status

Current state: *Discarded in deference to KIP-266*

Discussion thread:

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

Released: (planned 1.2)

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

There are two related WIP PRs:

- https://github.com/apache/kafka/pull/4861
- https://github.com/apache/kafka/pull/4855

Motivation

The current implementation of poll takes a timeout parameter, but the timeout doesn't apply to all parts of the implementation that block.

In a call to poll, there are basically two phases:

1. wait-for-metadata
2. poll-for-new-records

Currently, the timeout only applies to #2, and the "timeout" for #1 is infinite.

This causes a number of problems. First, several have noticed conditions in which wait-for-metadata blocks the thread uninterruptibly forever (see the linked Jiras). Second, there is a need for users of Consumer to just do #1, which they currently do by passing a timeout of 0ms, but the semantics of this usage are not clean (e.g., it's not clear whether or not it's safe to ignore the result). Third, during a regular poll, despite passing a specific timeout, there's no guarantee about how long the call will block, since an indefinite amount of time may be spent in phase #1.

Proposed Public Interface Change

We propose the following interface changes to Consumer and KafkaConsumer:

```
void waitForAssignment(long timeoutMs); // new method that just does current phase #1 of poll

ConsumerRecords<K, V> poll(long timeoutMs); // existing method, but timeout now applies to the whole operation
```

So waitForAssignment will be the new recommended method for those who just wish to block until the consumer gets an assignment, fetches offsets, etc. This method will throw a TimeoutException if it doesn't complete within the timeout.

And poll will begin enforcing the timeout on the entire operation, dividing the time between waiting for metadata and polling for records. This method will continue to return an empty collection if it doesn't complete phase #2 within the timeout. If phase #1 times out, poll will throw a TimeoutException.

Just as it does today, poll will still send async requests when the timeout is too low to actually wait for a response.

Compatibility, Deprecation, and Migration Plan

Since poll will send async requests and check on the responses in subsequent calls, it should continue to work as expected if callers retry their calls.

There is a possibility that callers depend on the consumer blocking until it gets an assignment, though, so we plan to throw a TimeoutException if phase #1 times out. This should be sufficient to alert callers to the change in semantics and protect them from broken assumptions.

Test Plan

This change will break numerous of our tests that call poll. We will update the tests either to allow more time to poll or to call the new method.

Also Qiang Zhao has submitted a new unit test verifying poll doesn't block forever: https://github.com/apache/kafka/pull/4861/files#diff-444ce0cab1dc8ec666294c4eafc9baefR1195

Rejected Alternatives

Just change the semantics of poll with no new method

This would be the simplest thing to do. We could just change the semantics of poll to use the timeout for both metadata and data. Adding about 30s to the timeouts in all our tests is sufficient to get them to pass, so we could share the same advice in the release notes.

But this actually breaks the existing use case for blocking on assignment, which we have witnessed in some open source projects. There would simply be no way to allow enough time for assignment, but not enough to to a poll, so you'd be forced to save the results for later processing. In other words, this alternative causes more disruption for some users, since it would force them to restructure their Consumer usage instead of just using a new method where appropriate.

Method variant with separate timeout for metadata

We could instead add a new poll variant like:

```
ConsumerRecords<K, V> poll(long metadataTimeoutMs, long pollTimeoutMs);
```

This would provide a more intuitive way to transition from the current system without changing any semantics, since the existing poll implementation is effectively

```
poll(Long.MAX_VALUE, timeoutMs)
```

But aside from the migration being easier, this variant is not that great. The parameters leak information about what is going on inside the method.

Plus, from an informal survey of usage, it really seems like you would have two distinct use cases, `poll(some_value, 0)` with the result ignored for just getting an assignment, and `poll(some_value, some_other_value)` with the result saved for really doing a poll. In the former usage, I'd argue it's actually a bit unsafe to ignore the result, since you can't be *sure* that poll won't return some results, and if it does, you'll miss them (the consumer would have no way of knowing that you're ignoring the results). The latter usage is also not great, since it's not clear how I as a caller should divide my available time between the assignment "phase" and actually polling.

Specify the metadata timeout via config

We could instead add a new config like one of these:

```
block.for.metadata := true|false
block.for.metadata.ms := [0,Long.MAX_VALUE]
```

This could let us allow poll to not block forever, while also by default not changing its semantics at all.

But on the downside, it means that those who really just want to block on metadata have to continue calling poll with a magical timeout of 0, and it's still not clear whether it's safe to ignore the result. And it means that callers who **do** specify a specific value for timeout will still have a hard time knowing how long the call will actually take.