KIP-945: Update threading model for Consumer

Status

Current state: *WIP*

Discussion thread: *here*

JIRA: *here*

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

Note this is a joint proposal by Philip Nee, Kirk True, and Lianet Magrans.

Motivation

This KIP documents the updated threading model of the `Consumer` implementation of the client.

The complexity of the consumer has increased and with it the code to support and fix bugs. Patches and hotfixes in the past years have heavily impacted the readability of the code. The complex code path and intertwined logic make the code difficult to modify and comprehend. Additionally, logic is at times executed on application threads and at other times on the dedicated, internal *heartbeat* thread. The asynchronous nature of the current implementation has lead to many bugs (which are labeled with the new-consumer-threading-should-fix label. The motivation is to simplify the structure of the code by clearly defining—and removing, where possible—the asynchronous code.

The simplification will also allow us to implement the necessary primitives for KIP-848.

Public Interfaces

This KIP has the explicit goal of making *no* changes to the public interfaces. The protocol, configuration, APIs, etc. will remain as they currently are. The internal behavior of the consumer is substantially changing and we want to ensure it is reviewed and vetted by the community.

Proposed Changes

Terminology

To help understand the design, we need to introduce some terminology. Terms designated with [1] apply to the current `KafkaConsumer` implementation and terms with [2] apply to the new implementation; a term may apply to both.

Term	Definition
Application event	A data structure specific to each `Consumer` API call that encapsulates application-provided data. For example, the application event specific to the `seek` event would include the user-provided topic information and offset. These events are enqueued onto the *application event queue* by the `Consumer`. Application events can optionally include a `Future` on which the *application thread* can issue a timed block, awaiting completion by the *ba ckground thread*.
Application event processor	Logic which processes *application events* on the *background thread*, interacting with the *request managers*.

Application event queue	A shared queue which stores *application events* enqueued by the *application thread*. These events are later dequeued by the *background thread* and given to the *application event processor* for execution.
Application thread	The thread that is executing the user's code that interacts with the `Consumer` API. Per the current implementation in `KafkaConsumer`, only one thread may call APIs at a time.
Background event	
Background event queue	A shared queue which stores *background events* enqueued by the *background thread*. The events are later dequeued by the *application thread* inside the `Consumer` and handled appropriately.
Background thread	An internal thread created for each `Consumer` instance on which the following operations are performed: • Execution of *application events* • Group membership • Managing network I/O requests and responses • Forwarding results to *application events* • Submitting *background events* for processing by the *application thread*
Event handler	Logic that pulls *events* from the *event queue* for processing on the *background thread*.
Heartbeat	Logic related to communicating liveness, group membership, etc. as introduced in KIP-62.
Network client delegate	
Request manager	An internal interface that is used by the background thread to handle the management of requested, inflight, and responded network I/O.

Threading Model

<TBD>

Background thread

<TBD>

Providing Data to the Background Thread

<TBD>

Getting Data from the Background Thread

<TBD>

Network I/O

<TBD>

Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?*
- *If we are changing behavior how will we phase out the older behavior?*
- *If we need special migration tools, describe them here.*
- *When will we remove the existing behavior?*

Test Plan

Describe in few sentences how the KIP will be tested. We are mostly interested in system tests (since unit-tests are specific to implementation details). How will we know that the implementation works as expected? How will we know nothing broke?

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.