

KIP-266: Fix consumer indefinite blocking behavior


- [Status](#)
- [Motivation](#)
- [Public Interfaces](#)
 - [Consumer#position](#)
 - [Consumer#committed](#) and [Consumer#commitSync](#)
 - [Consumer#poll](#)
 - [Consumer#partitionsFor](#)
 - [Consumer#listTopics](#)
 - [Consumer#offsetsForTimes](#)
 - [Consumer#beginningOffsets](#)
 - [Consumer#endOffsets](#)
 - [Consumer#close](#)
- [Compatibility, Deprecation, and Migration Plan](#)
- [Feasible and Rejected Alternatives](#)

Status

Current state: *Accepted*

Discussion thread: [here](#)

JIRA:

 Unable to render Jira issues macro, execution error.

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

Motivation

KafkaConsumer has a long history of indefinite blocking behavior which has been a continual cause of user frustration. This KIP aims to fix this problem in two ways:

1. We introduce a new configuration `default.api.timeout.ms` to control the maximum time that current blocking APIs will await before raising a timeout error.
2. We add overloaded APIs to allow for custom timeout control. This is intended for advanced usage such as in [Kafka Streams](#).

Some care must be taken in the case of the `poll()` since many applications depend on the current behavior of blocking until an assignment is found. This KIP retains this behavior for `poll(long)`, but introduces a new API `poll(Duration)`, which always returns when the timeout expires. We will deprecate `poll(long)` and remove it in a later major release.

Public Interfaces

This KIP adds `default.api.timeout.ms` as a new configuration for the consumer which controls the default timeout for methods which do not accept a timeout as an argument. The default value of `default.api.timeout.ms` will be one minute.

Below we document the APIs that this timeout will impact and how the behavior changes.

The following APIs currently block indefinitely until either the operation completes successfully or an unrecoverable error is encountered. Following this KIP, these methods will now raise `org.apache.kafka.common.errors.TimeoutException` if neither of these conditions have been reached prior to expiration of the time specified by `default.api.timeout.ms`.

```

void commitSync();

void commitSync(Map<TopicPartition, OffsetAndMetadata> offsets);

long position(TopicPartition partition);

OffsetAndMetadata committed(TopicPartition partition);

```

The following APIs currently block for at most the time configured by [request.timeout.ms](#) until the operation completed successfully or an unrecoverable error is encountered. Following this KIP, they will instead use the timeout indicated by [default.api.timeout.ms](#). As before, when the timeout is reached, `org.apache.kafka.common.errors.TimeoutException` will be raised to the user.

```

List<PartitionInfo> partitionsFor(String topic);

Map<String, List<PartitionInfo>> listTopics();

Map<TopicPartition, OffsetAndTimestamp> offsetsForTimes(Map<TopicPartition, Long> timestampsToSearch);

Map<TopicPartition, Long> beginningOffsets(Collection<TopicPartition> partitions);

Map<TopicPartition, Long> beginningOffsets(Collection<TopicPartition> partitions);

```

The current default timeout for the consumer is just over five minutes. It is intentionally set to a value higher than `max.poll.interval.ms`, which controls how long the rebalance can take and how long a `JoinGroup` request will be held in purgatory on the broker. In this KIP, we propose to change the default value of `request.timeout.ms` to 30 seconds. The `JoinGroup` API will be treated as a special case and its timeout will be set to a value derived from `max.poll.interval.ms`. All other request types will use the timeout configured by `request.timeout.ms`.

As mentioned above, this KIP also adds overloaded APIs to allow for custom timeout control. The new APIs are documented below:

Consumer#position

A `TimeoutException` will be thrown when the time spent exceeds `timeout`:

KafkaConsumer#position(TopicPartition topicPartition))

```
/**
 * Get the offset of the <i>next record</i> that will be fetched (if a record with that offset exists).
 * This method may issue a remote call to the server if there is no current position for the given
partition.
 * <p>
 * This call will block until either the position could be determined or an unrecoverable error is
 * encountered (in which case it is thrown to the caller).
 *
 * @param partition The partition to get the position for
+ * @param timeout    The maximum duration of the method
 *
 * @return The current position of the consumer (that is, the offset of the next record to be fetched)
 * @throws IllegalArgumentException if the provided TopicPartition is not assigned to this consumer
 * @throws org.apache.kafka.clients.consumer.InvalidOffsetException if no offset is currently defined for
 *         the partition
 * @throws org.apache.kafka.common.errors.WakeupException if {@link #wakeup()} is called before or while
this
 *         function is called
 * @throws org.apache.kafka.common.errors.InterruptException if the calling thread is interrupted before or
while
 *         this function is called
+ * @throws org.apache.kafka.common.errors.TimeoutException if time spent blocking exceeds the {@code
timeout}
 * @throws org.apache.kafka.common.errors.AuthenticationException if authentication fails. See the
exception for more details
 * @throws org.apache.kafka.common.errors.AuthorizationException if not authorized to the topic or to the
 *         configured groupId. See the exception for more details
 * @throws org.apache.kafka.common.KafkaException for any other unrecoverable errors
 */
+ long position(TopicPartition partition, Duration timeout);
```

Consumer#committed and Consumer#commitSync

Similarly, this will also be applied to other methods in KafkaConsumer that blocks indefinitely.

KafkaConsumer#blocking methods

```
/**
 * Get the last committed offset for the given partition (whether the commit happened by this process or
 * another). This offset will be used as the position for the consumer in the event of a failure.
 * <p>
 * This call will block to do a remote call to get the latest committed offsets from the server.
 *
 * @param partition The partition to check
+ * @param timeout    The maximum duration of the method
 *
 * @return The last committed offset and metadata or null if there was no prior commit
 * @throws org.apache.kafka.common.errors.WakeupException if {@link #wakeup()} is called before or while
this
 *         function is called
 * @throws org.apache.kafka.common.errors.InterruptException if the calling thread is interrupted before or
while
 *         this function is called
 * @throws org.apache.kafka.common.errors.AuthenticationException if authentication fails. See the
exception for more details
 * @throws org.apache.kafka.common.errors.AuthorizationException if not authorized to the topic or to the
 *         configured groupId. See the exception for more details
 * @throws org.apache.kafka.common.KafkaException for any other unrecoverable errors
+ * @throws org.apache.kafka.common.errors.TimeoutException if time spent blocking exceeds the {@code
timeout}
 */
OffsetAndMetadata committed(TopicPartition partition, final Duration timeout)'

/**
```

```

    * Commit the specified offsets for the specified list of topics and partitions.
    * <p>
    * This commits offsets to Kafka. The offsets committed using this API will be used on the first fetch
after every
    * rebalance and also on startup. As such, if you need to store offsets in anything other than Kafka, this
API
    * should not be used. The committed offset should be the next message your application will consume,
    * i.e. lastProcessedMessageOffset + 1.
    * <p>
    * This is a synchronous commits and will block until either the commit succeeds or an unrecoverable error
is
    * encountered (in which case it is thrown to the caller).
    * <p>
    * Note that asynchronous offset commits sent previously with the {@link #commitAsync(OffsetCommitCallback)}
    * (or similar) are guaranteed to have their callbacks invoked prior to completion of this method.
    *
    * @param offsets A map of offsets by partition with associated metadata
+    * @param timeout Maximum duration to block
    *
    * @throws org.apache.kafka.clients.consumer.CommitFailedException if the commit failed and cannot be
retrieved.
    *
    * This can only occur if you are using automatic group management with {@link #subscribe
(Collection)},
    *
    * or if there is an active group with the same groupId which is using group management.
    * @throws org.apache.kafka.common.errors.WakeupException if {@link #wakeup()} is called before or while
this
    *
    * function is called
    * @throws org.apache.kafka.common.errors.InterruptException if the calling thread is interrupted before or
while
    *
    * this function is called
+.    * @throws org.apache.kafka.common.errors.TimeoutException if time spent blocking exceeds the {@code
timeout}
    * @throws org.apache.kafka.common.errors.AuthenticationException if authentication fails. See the
exception for more details
    * @throws org.apache.kafka.common.errors.AuthorizationException if not authorized to the topic or to the
    *
    * configured groupId. See the exception for more details
    * @throws java.lang.IllegalArgumentException if the committed offset is negative
    * @throws org.apache.kafka.common.KafkaException for any other unrecoverable errors (e.g. if offset
metadata
    *
    * is too large or if the topic does not exist).
    */
    void commitSync(final Map<TopicPartition, OffsetAndMetadata> offsets, final Duration timeout);

/**
    * Commit offsets returned on the last {@link #poll(Duration) poll()} for all the subscribed list of topics
and
    * partitions.
    * <p>
    * This commits offsets only to Kafka. The offsets committed using this API will be used on the first fetch
after
    * every rebalance and also on startup. As such, if you need to store offsets in anything other than Kafka,
this API
    * should not be used.
    * <p>
    * This is a synchronous commits and will block until either the commit succeeds, an unrecoverable error is
    * encountered (in which case it is thrown to the caller), or the passed timeout expires.
    * <p>
    * Note that asynchronous offset commits sent previously with the {@link #commitAsync(OffsetCommitCallback)}
    * (or similar) are guaranteed to have their callbacks invoked prior to completion of this method.
    *
    * @param timeout Maximum duration to block
+    *
    * @throws org.apache.kafka.clients.consumer.CommitFailedException if the commit failed and cannot be
retrieved.
    *
    * This can only occur if you are using automatic group management with {@link #subscribe
(Collection)},
    *
    * or if there is an active group with the same groupId which is using group management.
    * @throws org.apache.kafka.common.errors.WakeupException if {@link #wakeup()} is called before or while
this
    *
    * function is called

```

```

    * @throws org.apache.kafka.common.errors.InterruptException if the calling thread is interrupted before or
while
    *
    * this function is called
    * @throws org.apache.kafka.common.errors.AuthenticationException if authentication fails. See the
exception for more details
    * @throws org.apache.kafka.common.errors.AuthorizationException if not authorized to the topic or to the
    * configured groupId. See the exception for more details
    * @throws org.apache.kafka.common.KafkaException for any other unrecoverable errors (e.g. if offset
metadata
    * is too large or if the topic does not exist).
+    * @throws org.apache.kafka.common.errors.TimeoutException if the timeout expires before successful
completion
    * of the offset commit
    */
@Override
public void commitSync(Duration timeout);

```

Currently, `commitSync` does not accept a user-provided timeout, but by default, will block indefinitely by setting wait time to `Long.MAX_VALUE`. To accomodate for a potential hanging block,

the new `KafkaConsumer#commitSync` will accept user-specified timeout.

Consumer#poll

The pre-existing variant `poll(long timeout)` would block indefinitely for metadata updates if they were needed, then it would issue a fetch and poll for `timeout` ms for new records. The initial indefinite metadata block caused applications to become stuck when the brokers became unavailable. The existence of the timeout parameter made the indefinite block especially unintuitive.

We will add a new method `poll(Duration timeout)` with the semantics:

1. if a metadata update is needed:
 - a. send (asynchronous) metadata requests
 - b. poll for metadata responses (counts against timeout)
 - if no response within timeout, return an empty collection immediately
2. if there is fetch data available, return it immediately
3. if there is no fetch request in flight, send fetch requests
4. poll for fetch responses (counts against timeout)
 - if no response within timeout, return an empty collection (leaving async fetch request for the next poll)
 - if we get a response, return the response

We will deprecate the original method, `poll(long timeout)`, and we will not change its semantics, so it remains:

1. if a metadata update is needed:
 - a. send (asynchronous) metadata requests
 - b. poll for metadata responses *indefinitely until we get it*
2. if there is fetch data available, return it immediately
3. if there is no fetch request in flight, send fetch requests
4. poll for fetch responses (counts against timeout)
 - if no response within timeout, return an empty collection (leaving async fetch request for the next poll)
 - if we get a response, return the response

One notable usage is prohibited by the new `poll`: previously, you could call `poll(0)` to block for metadata updates, for example to initialize the client, supposedly without fetching records. Note, though, that this behavior is not according to any contract, and there is no guarantee that `poll(0)` won't return records the first time it's called. Therefore, it has always been unsafe to ignore the response.

Note that `poll()` doesn't throw a `TimeoutException` because its async semantics are well defined. I.e., it is well defined to return an empty response when there's no data available, and it's designed to be called repeatedly to check for data (hence the name).

```

/**
 * Fetch data for the topics or partitions specified using one of the subscribe/assign APIs. It is an error to
 * not have
 * subscribed to any topics or partitions before polling for data.
 * <p>
 * On each poll, consumer will try to use the last consumed offset as the starting offset and fetch
 * sequentially. The last
 * consumed offset can be manually set through {@link #seek(TopicPartition, long)} or automatically set as the
 * last committed
 * offset for the subscribed list of partitions
 *
 *
 * @param timeout The maximum time to block and poll for metadata updates or data.
 *
 * @return map of topic to records since the last fetch for the subscribed list of topics and partitions
 *
 * @throws org.apache.kafka.clients.consumer.InvalidOffsetException if the offset for a partition or set of
 * partitions is undefined or out of range and no offset reset policy has been configured
 * @throws org.apache.kafka.common.errors.WakeupException if {@link #wakeup()} is called before or while this
 * function is called
 * @throws org.apache.kafka.common.errors.InterruptException if the calling thread is interrupted before or
 * while
 * this function is called
 * @throws org.apache.kafka.common.errors.AuthenticationException if authentication fails. See the exception
 * for more details
 * @throws org.apache.kafka.common.errors.AuthorizationException if caller lacks Read access to any of the
 * subscribed
 * topics or to the configured groupId. See the exception for more details
 * @throws org.apache.kafka.common.KafkaException for any other unrecoverable errors (e.g. invalid groupId or
 * session timeout, errors deserializing key/value pairs, or any new error cases in future versions)
 * @throws java.lang.IllegalArgumentException if the timeout value is negative
 * @throws java.lang.IllegalStateException if the consumer is not subscribed to any topics or manually assigned
 * any
 * partitions to consume from
 */
public ConsumerRecords<K, V> poll(final Duration timeout)

```

We will mark the existing `poll()` method as deprecated.

Consumer#partitionsFor

```

/**
 * Get metadata about the partitions for a given topic. This method will issue a remote call to the server if it
 * does not already have any metadata about the given topic.
 *
 * @param topic The topic to get partition metadata for
 * @param timeout The maximum time this operation will block
 *
 * @return The list of partitions
 * @throws org.apache.kafka.common.errors.TimeoutException if time spent blocking exceeds the {@code timeout}
 * @throws org.apache.kafka.common.errors.WakeupException if {@link #wakeup()} is called before or while this
 * function is called
 * @throws org.apache.kafka.common.errors.InterruptException if the calling thread is interrupted before or
 * while
 * this function is called
 * @throws org.apache.kafka.common.errors.AuthenticationException if authentication fails. See the exception
 * for more details
 * @throws org.apache.kafka.common.errors.AuthorizationException if not authorized to the specified topic. See
 * the exception for more details
 * @throws org.apache.kafka.common.errors.TimeoutException if the topic metadata could not be fetched before
 * expiration of the configured request timeout
 * @throws org.apache.kafka.common.KafkaException for any other unrecoverable errors
 */
List<PartitionInfo> partitionsFor(String topic, Duration timeout);

```

Consumer#listTopics

```
/**
 * Get metadata about partitions for all topics that the user is authorized to view. This method will issue a
 * remote call to the server.
 *
 * @param timeout The maximum time this operation will block
 *
 * @return The map of topics and its partitions
 * @throws org.apache.kafka.common.errors.TimeoutException if time spent blocking exceeds the {@code timeout}
 * @throws org.apache.kafka.common.errors.WakeupException if {@link #wakeup()} is called before or while this
 *         function is called
 * @throws org.apache.kafka.common.errors.InterruptException if the calling thread is interrupted before or
 *         while
 *         this function is called
 * @throws org.apache.kafka.common.errors.TimeoutException if the topic metadata could not be fetched before
 *         expiration of the configured request timeout
 * @throws org.apache.kafka.common.KafkaException for any other unrecoverable errors
 */
Map<String, List<PartitionInfo>> listTopics(Duration timeout)
```

Consumer#offsetsForTimes

```
/**
 * Look up the offsets for the given partitions by timestamp. The returned offset for each partition is the
 * earliest offset whose timestamp is greater than or equal to the given timestamp in the corresponding
 * partition.
 *
 * * This is a blocking call. The consumer does not have to be assigned the partitions.
 * * If the message format version in a partition is before 0.10.0, i.e. the messages do not have timestamps, null
 * will be returned for that partition.
 *
 * @param timestampsToSearch the mapping from partition to the timestamp to look up.
 * @param timeout The maximum time this operation will block
 *
 * @return a mapping from partition to the timestamp and offset of the first message with timestamp greater
 *         than or equal to the target timestamp. {@code null} will be returned for the partition if there is no
 *         such message.
 * @throws org.apache.kafka.common.errors.TimeoutException if time spent blocking exceeds the {@code timeout}
 * @throws org.apache.kafka.common.errors.AuthenticationException if authentication fails. See the exception
 *         for more details
 * @throws org.apache.kafka.common.errors.AuthorizationException if not authorized to the topic(s). See the
 *         exception for more details
 * @throws IllegalArgumentException if the target timestamp is negative
 * @throws org.apache.kafka.common.errors.TimeoutException if the offset metadata could not be fetched before
 *         expiration of the configured {@code request.timeout.ms}
 * @throws org.apache.kafka.common.errors.UnsupportedVersionException if the broker does not support looking up
 *         the offsets by timestamp
 */
Map<TopicPartition, OffsetAndTimestamp> offsetsForTimes(Map<TopicPartition, Long> timestampsToSearch, Duration
timeout)
```

Consumer#beginningOffsets

```

/**
 * Get the first offset for the given partitions.
 * <p>
 * This method does not change the current consumer position of the partitions.
 *
 * @see #seekToBeginning(Collection)
 *
 * @param partitions the partitions to get the earliest offsets.
 * @param timeout The maximum time this operation will block
 *
 * @return The earliest available offsets for the given partitions
 * @throws org.apache.kafka.common.errors.TimeoutException if time spent blocking exceeds the {@code timeout}
 * @throws org.apache.kafka.common.errors.AuthenticationException if authentication fails. See the exception
 for more details
 * @throws org.apache.kafka.common.errors.AuthorizationException if not authorized to the topic(s). See the
 exception for more details
 * @throws org.apache.kafka.common.errors.TimeoutException if the offsets could not be fetched before
 *         expiration of the configured {@code request.timeout.ms}
 */
Map<TopicPartition, Long> beginningOffsets(Collection<TopicPartition> partitions, Duration timeout)

```

Consumer#endOffsets

```

/**
 * Get the end offsets for the given partitions. In the default {@code read_uncommitted} isolation level, the
 end
 * offset is the high watermark (that is, the offset of the last successfully replicated message plus one). For
 * {@code read_committed} consumers, the end offset is the last stable offset (LSO), which is the minimum of
 * the high watermark and the smallest offset of any open transaction. Finally, if the partition has never been
 * written to, the end offset is 0.
 *
 * <p>
 * This method does not change the current consumer position of the partitions.
 *
 * @see #seekToEnd(Collection)
 *
 * @param partitions the partitions to get the end offsets.
 * @param timeout The maximum time this operation will block
 *
 * @return The end offsets for the given partitions.
 * @throws org.apache.kafka.common.errors.TimeoutException if time spent blocking exceeds the {@code timeout}
 * @throws org.apache.kafka.common.errors.AuthenticationException if authentication fails. See the exception
 for more details
 * @throws org.apache.kafka.common.errors.AuthorizationException if not authorized to the topic(s). See the
 exception for more details
 * @throws org.apache.kafka.common.errors.TimeoutException if the offsets could not be fetched before
 *         expiration of the configured {@code request.timeout.ms}
 */
Map<TopicPartition, Long> endOffsets(Collection<TopicPartition> partitions, Duration timeout)

```

Consumer#close

Notes:

- `close()` already is a variant with no parameters and applies a default from config. However, this variant will **not** be deprecated because it called by the `Closeable` interface.
- `close(long, TimeUnit)` also exists as a variant. This one **will** be deprecated in favor of the new `close(Duration)` variant for consistency
- The existing semantics of `close` is **not** to throw a `TimeoutException`. Instead, after waiting for the timeout, it forces itself closed.


```

/**
 * Tries to close the consumer cleanly within the specified timeout. This method waits up to
 * {@code timeout} for the consumer to complete pending commits and leave the group.
 * If auto-commit is enabled, this will commit the current offsets if possible within the
 * timeout. If the consumer is unable to complete offset commits and gracefully leave the group
 * before the timeout expires, the consumer is force closed. Note that {@link #wakeup()} cannot be
 * used to interrupt close.
 *
 * @param timeout The maximum time to wait for consumer to close gracefully. The value must be
 *                non-negative. Specifying a timeout of zero means do not wait for pending requests to complete.
 *
 * @throws IllegalArgumentException If the {@code timeout} is negative.
 * @throws InterruptedException If the thread is interrupted before or while this function is called
 * @throws org.apache.kafka.common.KafkaException for any other error during close
 */
public void close(Duration timeout)

```

Compatibility, Deprecation, and Migration Plan

Since old methods will not be modified, preexisting data frameworks will not be affected. However, some of these methods will be deprecated in favor of methods which are bound by a specific time limit.

The introduction of `default.api.timeout.ms` causes a slight change of behavior since some of the blocking APIs will now raise `TimeoutException` rather than their current blocking behavior. The change is compatible with the current API since `TimeoutException` is a `KafkaException`. Additionally, since `TimeoutException` is retrieable, any existing retry logic will work as expected.

Feasible and Rejected Alternatives

Please see [KIP-288](#) for other rejected alternatives.

In discussion, many have raised the idea of using a new config to set timeout time for methods which is being changed in this KIP. It would not be recommended to use one config for all methods. However, we could use it for similar methods (e.g. methods which call `updateFetchPositions()` will block using one timeout configured by the user). In this manner, we could incorporate both the config and the added timeout parameter into the code.

Another alternative of interest is that we should add a new overload for `poll()`, particularly since the changing the old method can become unwieldy between different Kafka versions. To elaborate, a `Timeout` parameter will also be added to the `poll()` overload.

One alternative was to add a `timeout` parameter to the current `position()` and other methods. However, the changes made by the user will be much more extensive then basing the time constraint on `requestTimeoutMs` because the method signature has been changed.

Another possibility was the usage of `requestTimeoutMs` to bound `position()`, however, this would make the method highly inflexible, especially since `requestTimeoutMs` is already being used by multiple other methods