KIP-286: producer.send() should not block on metadata update

Status

Current state: *Under Discussion*

Discussion thread: *here*

JIRA: *here*

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

Motivation

Currently producer.send() may block up to max.block.ms if metadata is not available. In general one of the following three outcome may happen if metadata is not available:

- producer.send() does block (if max.block.ms is 0) and user will drop message. However, this probably does not make sense to drop message if producer still has memory to buffer data (as configured by buffer.memory). In particular, when the producer is just instantiated, it is very likely that the metadata is not available when the producer.send() called for the first time, and we probably don't want to always drop the first few messages.

- producer.send() blocks user thread which can cause problem for latency sensitive front-end application.

- producer.send() blocks but user uses an extra thread and buffer to make sure that the user-thread does not block. This approach can work but requires non-trivial implementation on the user side to keep track of the the extra buffer to make sure it does not cause OOM etc.

It appears that none of the outcomes above for producer.send() is ideal if metadata is not available. This KIP proposes to improve the user experience and let producer.send() not block on metadata update.

Public Interfaces

We will change the behavior of producer.send() so that it does not block on metadata update. If metadata is not available, producer.send() will enqueue the request into the producer's buffer and return a future immediately to user. If the queue is already full, producer.send() will still block up to max.block.ms as does the existing Kafka producer's implementation.

Proposed Changes

Before the proposed change of this KIP:

The following will happen in the user thread after producer.send() is called by the user thread:

1) Get partition information of the give topic to determine the partition for the given message. Wait up to max.block.ms for cluster metadata update if metadata is not available for the give topic.
2) Serialize key and value of the given message.
3) Maybe allocate memory to create a new batch for the corresponding partition of the message. Wait up to max.block.ms if there is not enough free memory yet.
4) Compress and put the message into the batch.
5) proucer.send() will unblock and return a future.

After the proposed change of this KIP:

The following will happen in the user thread after producer.send() is called by the user thread:

1) Serialize key and value of the given message.

2.1) If the partition information of the topic is not available:
- Allocate memory to store the given uncompressed message. Wait up to max.block.ms if there is not enough free memory yet.
- Put the message into a per-topic queue.

2.2) If the partition information of the topic is available:
- Maybe allocate memory to create a new batch for the corresponding partition of the message. Wait up to max.block.ms if there is not enough free memory yet.
- Compress and put the message into the batch.

3) For every message in every non-empty per-topic queue whose topic's partition information is available,
- Maybe allocate memory to create a new batch for the corresponding partition of the message. Wait up to max.block.ms if there is not enough free memory yet.
- Compress and put the message into the batch.
- Dealloate the buffer for the message

4) proucer.send() will unblock and return a future.

In the rare scenario where user thread sends a message when the metadata is not available, but never calls producer.send() after metadata is available, the message will never be put into the per-partition queue with the changes above. In order to take care of this scenario, this KIP should additional modify the producer's IO thread such that, if no new ProduceRequest can be generated and there exists non-empty per-topic queue whose topic's partition information is available, IO thread should do the step 3) above to move these messages into the per-partition queue.

Theoretical Performance Analysis

Here we analyze how the changes proposed in this KIP would change the performance of producer:

1) Worst case time of producer.send() suppose the producer queue is not full.

Before this KIP, the producer.send() may block up to max.block.ms for metadata to become available. This KIP improves the latency in this scenario by putting the message in the per-topic queue without blocking producer.send().

This KIP may also increase the time of producer.send() in the following scenario. The first time producer.send() is called after metadata becomes available, according to step 3) in the proposed changes above, producer.send() will compress messages and put these messages into the batch for the corresponding partitions. If there is a lot of messages in the per-topic queues, this can take some time. But amount of time to compress messages should be considerably less than the time needed to wait for metadata (and compress messages) because user threads are better utilized by not waiting for metadata.

2) Throughput

This change does not affect the rate at which IO thread can send messages to producer. Thus the difference in throughput should mainly be caused by how we do computation intensive tasks (i.e. serialization and compression) and how we wait for metadata.

If producer.send() is called infrequently, throughput should not be an issue. If producer.send() is called frequently, all computation intensive task, i.e. compression and serialization, will still be done by the user threads, which is the same as the existing producer implementation. Thus the change proposed in the KIP sohuld not reduce throughput terms of computation.

On the other hand, since user thread will not longer block on metadata, user thread can be better utilized to do other useful work. So the overall throughput should be better with the change proposed in this KIP.

3) Latency of message from the time producer.send() is called to the time the message is acknowledged

If metadata is available, the latency should be the same.

If metadata is not available, messages can not be sent. latency should be dominated by the time to wait for metadata and will still be the same. Once the metadata becomes available, user thread will be busy compressing these messages and putting them into the per-partition queue. Since the throughput should be better (see the analysis above), the latency in this case should also be slightly better.

Compatibility, Deprecation, and Migration Plan

There is not compatibility concern or migration plan needed for this change.

Rejected alternative

- Make producer.send() non-blocking even if the queue is full.

Some user may still want to wait for a configurable amount of time on producer.send() if the queue is full instead of dropping messages immedidately.
Users who want complete non-blocking producer.send() can set max.block.ms to 0.