KIP-185: Make exactly once in order delivery per partition the default producer setting

Status

Current state: Under Discussion

Discussion thread: here

JIRA: KAFKA-5795 - Getting issue details... STATUS

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

Motivation

At the moment, the default settings of the Kafka Producer provide at-most once delivery without ordering guarantees per partition. This is the weakest delivery guarantee Kafka provides. Since Kafka 0.11.0, Kafka's strongest delivery guarantee is exactly once, in order, delivery per partition.
The next release of Kafka is 1.0.0, and it seems like a good time to have Kafka ship with its strongest delivery guarantees as the default.

Background

Let's clarify the terminology we use in the rest of this document to ensure we are all on the same page.

Definitions

Here is how we define the various delivery semantics.

At-most once: Acknowledged messages would appear in the log at most once. In other words some acknowledged messages may be lost.

At-least once: Every acknowledged message appears in the log at least once, ie. there may be duplicates introduced due to internal retries.

Exactly-once: Every acknowledged message appears in the log exactly once, without duplicates.

Failure modes

We distinguish between two kinds of failures:

Transient failures: Any failure from which a host can recover to full functionality after the cause has passed. For instance, power failures, network interrupts, buggy code, etc.

Hard failures: Failures from which there is no way for a host to recover. For instance, disk failure, data corruption due to a bug, etc.

The guarantees defined above only apply to transient failures. If data is lost or the log becomes un-readable, acknowledged messages may be lost regardless of the originally promised semantics.

Concurrent transient failures may also result in the violation of at-least once and exactly-once delivery semantics.

Replication and delivery guarantees

By he definitions above, if a topic is configured to have only a single replica, the best guarantee that Kafka can provide is at-most once delivery, regardless of the producer settings.

The core reason is that the flush to disk happens asynchronously. So it may happen that there is a power failure --or the kafka process receives a `SIGKILL --` after acknowledging the message but before the flush happens. Thus acknowledged messages may be lost even due to a transient failure.

For the same reason, a concurrent transient failure may also result in acknowledged messages being lost. For instance, for a topic with replication-factor=3, if all three replicas suffer a simultaneous power outage after acknowledging the message but before flushing the data to disk, we would lose the acknowledged messages.

Public Interfaces

We would like to change the default values for the following configurations of the Kafka Producer.

config	current default value	proposed default value
enable.idempotence	false	true
acks	1	all
max.in.flight.requests.per.connection	5	2
retries	0	MAX_INT

There will be no other publicly visible changes required.

The changes above would guarantee exactly once, in order delivery per partition for topics with `replication-factor >= 2`, and assuming that the system doesn't suffer multiple hard failures or concurrent transient failures.

Proposed Changes

Code changes

Currently, the idempotent producer requires max.in.flight.requests.per.connection=1 for correctness reasons. We will need to make client and broker changes to support max.in.flight.requests.per.connection > 1 with the idempotent producer. The details of the changes required are in this ticket.

We would also need the changes described in these documents:

1. Kafka Exactly Once - Solving the problem of spurious OutOfOrderSequence errors
2. Kafka Exactly Once - Dealing with older message formats when idempotence is enabled

Performance considerations

Next we will delineate the reasons for choosing the proposed values for the configurations in question and the performance impact of the same. A summary of the performance tests we ran to understand the impact of these changes can be found here: An analysis of the impact of max.in.flight.requests.per.connection and acks on Producer performance.

For the same value of max.in.flight.requests, and acks, we have observed that enabling idempotence alone does not significantly impact performance. Detailed results supporting this statement are available here.

Why change max.in.flight.requests.per.connection from 5 to 2?

The max.in.flight.requests.per.connection setting was introduced to improve producer throughput by achieving better pipelining: instead of waiting for the response from the broker before sending the next batch of records, the producer could keep the broker's request queue full by having multiple requests in flight. This removes the dead time on the broker from when it sends a response to the time it receives the next request. However, the cost is that we could have out of order writes when there are failures.
With the changes proposed in KAFKA-5494, we can still have ordering guarantees with max.in.flight > 1 thanks to the sequence numbers introduced in KIP-98.
Further, the results above show that there is a large improvement in throughput and latency when we go from max.in.flight=1 to max.in.flight=2, but then there no discernible difference for higher values of this setting. Hence we propose changing the default to 2.

Why change retries from 0 to MAX_INT?

The retries config was defaulted to 0 to ensure that internal producer retries don't introduce duplicates. With the idempotent producer introduced in KIP-98 - Exactly Once Delivery and Transactional Messaging, internal producer retries can no longer introduce duplicates. Hence, with the current proposal to enable the idempotent producer by default, we should let the producer retry as much as possible since there is no correctness penalty for doing so.

Why change acks from 1 to all?

With acks=1, we only have an at most once delivery guarantee. In particular, with acks=1, the broker could crash after acknowledging a message but before replicating it. This results in an acknowledged message being lost. In other words, if we want exactly once delivery, we need acks=all along with enable.idempotence=true.

The performance analysis above shows that there is an impact to latency when moving from acks=1 to acks=all. This is not entirely surprising: a ProduceRequest with acks=all is blocked on more RPCs since the followers need to fetch the newly appended data before the request is acknowledged, and hence we expect a hit to latency. Nonetheless, further analysis is ongoing to figure out the exact impact across different workloads and if there is a way to improve the situation.

Regardless, we believe strong durability guarantees out of the box are worth the cost of increased latency.

Compatibility, Deprecation, and Migration Plan

This KIP only proposes changes to the default settings of some client configurations. Hence no migration plans are required.

Further, we don't propose to deprecate any configurations at this point.

As for compatibility, it is possible that the change in defaults might cause unexpected changes in behavior for users who upgrade from older versions. We will call this out the release notes for the 1.0.0 release. Some of the differences are listed below.

Applications may receive the new OutOfOrderSequenceException

This exception indicates that previously acknowledged data was lost. With the new exactly once features we can detect this and report the error to the user. A properly configured producer and broker should only receive this exception for real data loss.

Producer performance profile may change

The acks=all setting would result worse througput and latency, as documented here.

Rejected Alternatives

This is a proposal to change the default semantics of the Kafka Producer. The specific values of the various variables are either necessary to achieve stronger semantics, or have been chosen through empirical observation. Hence there are no rejected alternatives at this point.