KIP-192 : Provide cleaner semantics when idempotence is enabled

Status

Current state: Accepted

Discussion thread: http://search-hadoop.com/m/Kafka/uyzND1DqvOK1jiI281?subj=+DISCUSS+KIP+192+Provide+cleaner+semantics+when+idempotence+is+enabled

JIRA: https://issues.apache.org/jira/browse/KAFKA-5793 and https://issues.apache.org/jira/browse/KAFKA-5794

Release Version *:* Update to the `RecordMetadata` class and `ProduceResponse` are in 1.0.0. The new values for `enable.idempotence` in the `ProducerConfig` and the updates to the `TopicMetadata` in `MetadataResponse` has been postponed to a future release.

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

Motivation

There are currently two situations where the behavior of the producer with idempotence enabled is less than satisfactory:

1. Currently the OutOfOrderSequence exception may be raised spuriously, for instance, if the producer state was removed on the server due to segments which are older than the retention limit being deleted. We would like the OutOfOrderSequence exception to unequivocally indicate data loss, and hence need to detect and handle these false positives.
2. There is no graceful way to handle enabling idempotence on the producer, and yet having some topics being on an older (pre 0.11.0) message format. This means that making idempotence the default is impossible, as the upgrade steps would simply not work. Hence we would like to introduce a 'safe' mode for idempotence where it will only be enabled if the underlying topic has the requisite message format.

Background

The two problems described above are detailed in the following pages, along with proposed solutions.

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

Public Interfaces

RecordMetadata (in 1.0.0)

With the changes in Kafka Exactly Once - Solving the problem of spurious OutOfOrderSequence errors, the broker may return a new `DUPLICATE_SEQUENCE` error code in some cases where a duplicate is detected but the metadata for the existing batch isn't cached in memory. When the producer receives this error, it is considered a success, but will not have the offset and timestamp information for the appended records. To help identify this state, we add `hasOffset` and add `hasTimestamp` methods to the `RecordMetadata`.

```
package org.apache.clients.producer;

public final class RecordMetadata {

  /**
   * Indicates whether the record metadata includes the offset.
   * @return true if the offset is available, false otherwise.
   */
  public boolean hasOffset();

  /**
   * Indicates whether the record metadata includes the timestamp.
   * @return true if the timestamp is available, false otherwise.
   */
  public boolean hasTimestamp();

}
```

TopicMetadataResponse

We add a 'MessageFormatVersion' field to the `TopicMetadata` returned in the `MetadataResponse`. This is used to selectively enable idempotence in `requested` mode when the partition actually supports it. See Kafka Exactly Once - Dealing with older message formats when idempotence is enabled for a description of precisely how this will be used.

We will also add the `MaxMessageBytes` topic config to the topic metadata response as part of these changes. While this is not required to enable any superior functionality in the idempotent producer, it would be useful to have this bit of metadata in the producer for future features, and hence we add it here so that we don't have to change the protocol again for such a minor field.

```
// TopicMetadataV3

TopicMetadata => TopicErrorCode
                 Topic
                 IsInternal
                                MessageFormatVersion
                                MaxMessageBytes
                                [PartitionMetadata]
 TopicErrorCode => int16
 Topic => String
 IsInternal => Boolean
 MessageFormatVersion => int8 (NEW)
 MaxMessageBytes => int32 (NEW)
 PartitionMetadata => PartitionMetadataV2
```

ProduceResponse (in 1.0.0)

We add a `logStartOffset` field to the produce response to help the producer identify when producer state has been lost due to retention time elapsing. See Kafka Exactly Once - Solving the problem of spurious OutOfOrderSequence errors for a precise description of how this will be used.

```
// ProduceResponse v4
ProduceResponse => [TopicName [Partition ErrorCode Offset Timestamp logStartOffset]]
                   ThrottleTime
 TopicName => string
 Partition => int32
 ErrorCode => int16
 Offset => int64
 Timestamp => int64
 ThrottleTime => int32
 logStartOffset => int64 (NEW)
```

Producer config changes

We introduce new values for the `enable.idempotence` configuration: `requested`, `required`, `off`.

Compatibility, Deprecation, and Migration Plan

For the Produce Request/Response updates, we follow the existing conventions for maintaining backward compatibility. New producers will continue to talk with old brokers using the old versions of the protocol.

The legacy values for `enable.idempotence` will be interpreted as follows by the new producer: `true` will mean `required`, `false` will mean `off.`

As part of these changes, we will deprecate the `true` and `false` options for `enable.idempotence` by logging a warning if these are used.

For applications which care about always receiving the offset and timestamp of produced records, there is a greater chance that these will not be available when idempotence is enabled (for instance a broker bounce would lose some of the cached record metadata, and if an internal producer retry resulted in a duplicate of a record which is dropped from the cache, the record metadata would not be returned for this record, even though the append is successful). Such applications should now check the new `RecordMetadata.hasOffset` and `RecordMetadtata.hasTimestamp` methods before using the values returned from the `RecordMetadata.offset` and `RecordMetadata.timestamp` methods.

Rejected Alternatives

This KIP contains changes to fix existing problems or clarify existing behavior. As such, there are not too many options for making these improvements within the existing solutions.