KIP-588: Allow producers to recover gracefully from transaction timeouts

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

Transactional coordinator uses producer epoch to guarantee a single writer scope for each transactional id. When a new producer with the same transactional id starts up and talks to the coordinator, the coordinator will bump the epoch. Thus when the older producer attempts to make progress, it shall be fenced by a fatal `ProducerFenced` exception as it still uses old epoch.

However, producer epoch could also be bumped when a transaction times out on the coordinator side. This could be caused by a short period of inactivity of the client due to network partition or long GC. When the producer goes back online and attempts to proceed, it will receive the exact `ProducerFenced` even though a conflicting producer doesn't exist. The application has to shut down the current producer and rebuild a new one instead, by placing an extra try-catch logic which is cumbersome.

We can improve this with the new APIs from KIP-360. When the coordinator times out a transaction, it can remember that fact and allow the existing producer to claim the bumped epoch and continue. We should also improve on the exception matching, specifically INVALID_PRODUCER_EPOCH is still overloaded. The error code should have its own dedicated exception type for clear handling.

Proposed Changes

Retry Workflow

1. When a transaction times out, set lastProducerEpoch to the current epoch and do the normal bump.

2. Any transactional requests from the old epoch result in a new TRANSACTION_TIMED_OUT error code, which is propagated to the application. This mechanism applies to all producer transaction coordinator APIs:

- AddPartitionsToTransaction
- AddOffsetsToTransaction
- EndTransaction

3. The producer recovers by internally sending InitProducerId with the current epoch. The coordinator returns the bumped epoch.

One extra issue that needs to be addressed is how to handle INVALID_PRODUCER_EPOCH from Produce requests. Partition leaders will not generally know if a bumped epoch was the result of a timed out transaction or a fenced producer. The underlying exception type is ProducerFenced which is also misleading, since only the transaction coordinator should be aware of whether a producer gets fenced, not the partition leader.

Based on these observations, a separate error code will be created as PRODUCE_FENCED, such that INVALID_PRODUCER_EPOCH is no longer fatal and could trigger KIP-360 logic. New producers will treat INVALID_PRODUCER_EPOCH as abortable when they come from Produce responses. In the next step, Producer would try to abort the transaction to detect whether this was due to a timeout or otherwise, as end transaction call shall also be protected by the new transaction timeout retry logic.

In essence, INVALID_PRODUCER_EPOCH should only get thrown from Produce requests and is retriable, while all transaction coordinator interacting protocols use PRODUCER_FENCED.

Old Brokers Error Propagation

When the client is on the latest version but the broker is old, the client shall still see INVALID_PRODUCER_EPOCH from transactional responses. This error should still be treated as fatal exception, and we don't need to check for broker version explicitly since new transaction coordinator doesn't reply with this error code anymore to new client. The new client would translate this as PRODUCER_FENCED when transiting to fatal error state.

Interaction With Older Clients

New transaction coordinator will still return INVALID_PRODUCER_EPOCH for older clients as this is the only error code they would recognize. In the same time, new partition leader will always return INVALID_PRODUCER_EPOCH as well which has no impact to older clients.

Public Interfaces

We will add a new retriable error code to allow producer distinguish a fatal fencing vs a soft retry after server side timeout:

```
TRANSACTION_TIMED_OUT(90, "The last ongoing transaction timed out on the coordinator, should retry
initialization with current epoch", TransactionTimedOutException::new);
```

The ProducerFencedException will be given a dedicated error code, and a new exception type shall be created for INVALID_PRODUCER_EPOCH :

```
INVALID_PRODUCER_EPOCH(47, "Producer attempted to produce with an old epoch.", InvalidProducerEpochException::
new), // modified
...
PRODUCER_FENCED(91, "There is a newer producer with the same transactionalId", ProducerFencedException::new);
```

To be able to recognize clients that are capable of handling new error codes, we need to bump a set of RPC protocols version by 1, to be specific:

1. AddPartitionsToTransaction to v2
2. AddOffsetsToTransaction to v2
3. EndTransaction to v2
4. InitProducerId to v4

On the public Producer API, we shall also document the exceptions to let user catch it, do the transaction abortion and restart a new transaction for both TransactionTimedOutException and InvalidProducerEpochException. The only exception is the "abortTransaction", which was supposed to be safe when being called inside the catch block for abortable exceptions.

KafkaProducer.java

```
/**
 * @throws TransactionTimedOutException if the producer has encountered a previously aborted transaction on
coordinator side.
 *         Application should catch it and retry starting another transaction in this case.
 * @throws org.apache.kafka.common.errors.InvalidProducerEpochException if the producer has attempted to
produce with an old epoch
 *         to the partition leader. See the exception for more details
 */
public void sendOffsetsToTransaction(Map<TopicPartition, OffsetAndMetadata> offsets, String consumerGroupId);

/**
 * @throws TransactionTimedOutException if the producer has encountered a previously aborted transaction on
coordinator side.
 *         Application should catch it and retry starting another transaction in this case.
 * @throws org.apache.kafka.common.errors.InvalidProducerEpochException if the producer has attempted to
produce with an old epoch
 *         to the partition leader. See the exception for more details
 */
public void sendOffsetsToTransaction(Map<TopicPartition, OffsetAndMetadata> offsets, ConsumerGroupMetadata
groupMetadata);

/**
 * @throws TransactionTimedOutException if the producer has encountered a previously aborted transaction on
coordinator side.
 *         Application should catch it and retry starting another transaction in this case.
 * @throws org.apache.kafka.common.errors.InvalidProducerEpochException if the producer has attempted to
produce with an old epoch
 *         to the partition leader. See the exception for more details
 */
public void commitTransaction();

/**
 * (In callback:)
 * @throws InvalidProducerEpochException if the producer has attempted to produce with an old epoch to the
partition leader.
 *         Application should catch it and retry starting another transaction in this case.
 */
public Future<RecordMetadata> send(ProducerRecord<K, V> record, Callback callback);
```

Code Example

A sample template of the new exception handling is shown below. Basically for non critical exceptions, just re-initializing the producer is fine. For critical ones, the producer has to be closed.

Sample.java

```
producer.initTransactions();
volatile boolean isRunning = true;

while (isRunning) {
        try {
            producer.beginTransaction();
                producer.send(new ProducerRecord<>("topic", new byte[1], new byte[1]), (metadata, exception) ->
{
                        // ProducerFenced should no longer be thrown from producer callback
            if (exception instanceOf OutOfOrderSequenceException ||
                                exception instanceOf AuthorizationException) {
                isRunning = false;
            }
                        // Other non-fatal exceptions will be handled internally through initPID retry
                );
            producer.commitTransaction();
         } catch (ProducerFencedException | OutOfOrderSequenceException | AuthorizationException e) {
                  // We can't recover from these exceptions, so our only option is to close the producer and
exit.
        break;
        } catch (KafkaException e) {
             // For all other exceptions, just abort the transaction and try again.
             producer.abortTransaction();
         }
}

producer.close();
```

Compatibility, Deprecation, and Migration Plan

To be able to benefit from this feature, user only needs to upgrade clients and brokers to the latest version, without ordering requirement. Compatibility story is already discussed in the previous section.

Rejected Alternatives

For the INVALID_PRODUCER_EPOCH, we could also consider keeping the current behavior still and let user do the abort transactions manually when catching the exception. This seems to be unnecessary and has more cost to educate users, thus we propose to let Producer handle this scenario internally.