KIP-360: Improve reliability of idempotent/transactional producer

Status

Current state: Adopted (2.5.0)

Discussion thread: here

JIRA:

KAFKA-8710 - Getting issue details... STATUS

KAFKA-8805 *- Getting issue details...* ***STATUS***

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

Motivation

Idempotent/transactional semantics depend on the broker retaining state for each active producer id (e.g. epoch and sequence number). When the broker loses that state–due to segment deletion or a call to DeleteRecords–then additional produce requests will result in the UNKNOWN_PRODUCER_ID error.

Currently the producer attempts to handle this error by comparing the last acknowledged offset with the log start offset from the produce response with the error. If the last acknowledged offset is smaller than the log start offset, then the producer assumes that the error is spurious. It resets the sequence number to 0 and retries using the existing epoch.

There are two main problems with this approach:

1. Resetting the sequence number is safe only if we are guaranteed that the produce request was not written to the log. If we had to retry the request–e.g. due to a disconnect–then we have no way to know whether the first attempt was successful or not. The only option at the moment is to treat this as a fatal error for the producer.
2. Currently there are not any strict guarantees that the log start offset will increase monotonically. In particular, a new leader can be elected without having seen the latest log start offset from the previous leader. The impact for the producer is that state that was once lost may resurface after a leader failover. This can lead to an out of sequence error when the producer has already reset its sequence number. This is a fatal error for the producer.

Resetting the sequence number is fundamentally unsafe because it violates the uniqueness of produced records. Additionally, the lack of validation on the first write of a producer introduces the possibility of non-monotonic updates and hence, dangling transactions. In this KIP, we propose to address these problems so 1) this error condition becomes rare, and 2) it is no longer fatal. For transactional producers, it will be possible to simply abort the current transaction and continue. We also make some effort to simplify error handling in the producer.

Proposed Changes

Our proposal has three parts: 1) safe epoch incrementing, 2) prolonged producer state retention, and 3) simplified client error handling.

Safe Epoch Incrementing: When the producer receives an UNKNOWN_PRODUCER_ID error, in addition to resetting the sequence number, we propose to bump the epoch. For the idempotent producer, bumping the epoch can be done locally since its producer id is unique. The gap at the moment is a safe way for the transactional producer to do so. The basic problem is that the producer may have already been fenced by another instance, so we do not want to allow it to continue.

We propose to alter the InitProducerId API to accept an optional current epoch and an optional current producerId. When provided, the transaction coordinator will verify that the provided epoch matches the current epoch and only allow the version bump if it does.

To simplify the handling, the producer will take the following steps upon receiving the UNKNOWN_PRODUCER_ID error:

1. Enter the ABORTABLE_ERROR state. The only way to recover is for the user to abort the current transaction.
2. Use the InitProducerId API to request an epoch bump using the current epoch.
3. If another producer has already bumped the epoch, this will result in a fatal INVALID_PRODUCER_EPOCH error.
4. If the epoch bump succeeds, the producer will reset sequence numbers back to 0 and continue after the next transaction begins.

Of course the producer may fail to receive the response from the InitProducerId call, so we need to make this API safe for retries. In the worst case, a retry may span coordinator failover, so we need to record in the transaction log whether the bump was the result of a new producer instance or not. We propose to add a new field to the transaction state message for the last epoch that was assigned to a producer instance. We also need to handle retries of a request that triggered the generation of a new producerId due to epoch exhaustion, so we propose to add a new field to the transaction state message for the previous producerId associated with the transaction. When the coordinator receives a new InitProducerId request, we will use the following logic to update the epoch:

1. If no producerId/epoch is provided, the producer is initializing for the first time.
    a. If there is no current producerId/epoch, generate a new producerId and set epoch=0.
    b. Otherwise, we need to bump the epoch, which could mean overflow:
        i. No overflow: Bump the epoch and return the current producerId with the bumped epoch. The last producerId/epoch will be set to empty.
        ii. Overflow: Initialize a new producerId with epoch=0 and return it. The last producerId/epoch will be set to empty.
2. If producerId/epoch is provided, then the producer is re-initializing after a failure. There are three cases:
    a. The provided producerId/epoch matches the existing producerId/epoch, so we need to bump the epoch. As above, we may need to generate a new producerId if there would be overflow bumping the epoch.
        i. No overflow: bump the epoch and return the current producerId with the bumped epoch. The last producerId/epoch will be set to the previous producerId/epoch.
        ii. Overflow: generate a new producerId with epoch=0. The last producerId/epoch will be set to the old producerId/epoch.
    b. The provided producerId/epoch matches the last producerId/epoch. The current producerId/epoch will be returned.
    c. Else return INVALID_PRODUCER_EPOCH

Another case we want to handle is InvalidProducerIdMapping. This error occurs following expiration of the producerId. It's possible that another producerId has been installed in its place following expiration (if another producer instance has become active), or the mapping is empty. We can safely retry the InitProducerId with the logic in this KIP in order to detect which case it is:

1. After receiving INVALID_PRODUCER_ID_MAPPING, the producer can send InitProducerId using the current producerId and epoch.
2. If no mapping exists, the coordinator can generate a new producerId and return it. If a transaction is in progress on the client, it will have to be aborted, but the producer can continue afterwards.
3. Otherwise if a different producerId has been assigned, then we can return INVALID_PRODUCER_EPOCH since that is effectively what has happened. This is intended to simplify error handling. The point is that there is a newer producer and it doesn't matter whether it has the same producer id or not.

Prolonged producer state retention: As proposed in KAFKA-7190, we will alter the behavior of the broker to retain the cached producer state even after it has been removed from the log. Previously we attempted to keep the producer state consistent with the the contents of the log so that we could rebuild it from the log if needed. However, it is rarely necessary to rebuild producer state, and it is more useful to retain the state we have as long as possible. Here we propose to remove it only when the transactional id expiration time has passed.

Note that it is possible for some disagreement on the current producer state between the replicas. One example is following a partition reassignment. A new replica will only see the producers which have state in the log, so this may lead to an unexpected UNKNOWN_PRODUCER_ID error if the new replica becomes a leader before any additional writes occur for that producer id. However, any inconsistency between the leader and follower about producer state does not cause any problems from the perspective of replication because followers always assume the leader's state is correct. Furthermore, since we will now have a way to safely bump the epoch on the producer when we see UNKNOWN_PRODUCER_ID, these edge cases are not fatal for the producer.

Simplified error handling: Much of the complexity in the error handling of the idempotent/transactional producer is a result of the UNKNOWN_PRODUCER_ID case. Since we are proposing to cache producer state for as long as the transactional id expiration time even after removal from the log, this should become a rare error, so we propose to simplify our handling of it. The current handling attempts to reason about the log start offset and whether or not the batch had been previously retried. If we are sure it is safe, then we attempt to adjust the sequence number of the failed request (and any inflight requests which followed). Not only is this behavior complex to implement, but continuing with subsequent batches introduces the potential for duplicates. Currently there is no easy way to prevent this from happening.

We propose the following simplifications:

1. Assignment of the epoch/sequence number to a record batch is permanent and happens at the time of the record send. We will remove the logic to adjust sequence numbers after failing a batch.
2. When we encounter a fatal error for a batch, we will fail all subsequent batches for that partition which have been assigned a sequence number.

Records will be guaranteed to be delivered in order up until the first fatal error and there will be no duplicates. For the transactional producer, the user can proceed by aborting the current transaction and ordering can still be guaranteed going forward. Internally, the producer will bump the epoch and reset sequence numbers for the next transaction. For the idempotent producer, the user can choose to fail or they can continue (with the possibility of duplication or reordering). If the user continues, the epoch will be bumped locally and the sequence number will be reset.

Public Interfaces

We will bump the InitProducerId API. The new schemas are provided below:

```
InitProducerIdRequest => TransactionalId TransactionTimeoutMs ProducerId Epoch
 TransactionalId => NULLABLE_STRING
 TransactionTimeoutMs => INT32
 ProducerId => INT64           // NEW
 Epoch => INT16                // NEW

InitProducerIdResponse => Error ProducerId Epoch
 Error => INT16
 ProducerId => INT64
 Epoch => INT16
```

The producerId in the request is used to disambiguate requests following expiration of the transactionalId. After a transactional id has expired, its state is removed from the log. If the id is used again in the future, a new producerId will be generated. For a producer which is being initialized for the first time, the producerId and epoch will be set to -1. For a producer which is reinitializing, a positive valued producerId and epoch must be provided. If either producerId or epoch is initialized to -1 and the other is not, the broker will return the INVALID_REQUEST error code.

As mentioned above, we will bump the version of the transaction state message to include the last epoch.

```
Value => Version ProducerId CurrentEpoch LastBump TxnTimeoutDuration TxnStatus [TxnPartitions]
TxnEntryLastUpdateTime TxnStartTime
  Version => 1 (INT16)
  ProducerId => INT64
  LastProducerId => INT64  // NEW
  CurrentEpoch => INT16
  LastEpoch => INT16  // NEW
  TxnTimeoutDuration => INT32
  TxnStatus => INT8
  TxnPartitions => [Topic [Partition]]
     Topic => STRING
     Partition => INT32
  TxnLastUpdateTime => INT64
  TxnStartTime => INT64
```

As described above, the last epoch is initialized based on the epoch provided in the InitProducerId call. For a new producer instance, the value will be -1. The last producer id is the previous producer ID associated with the transaction. For a new producer instance, the value will be -1.

Compatibility, Deprecation, and Migration Plan

The main problem from a compatibility perspective is dealing with the existing producers which reset the sequence number to 0 but continue to use the same epoch. We believe that caching the producer state even after it is no longer retained in the log will make the UNKNOWN_PRODUCER_ID error unlikely in practice, so this resetting logic should be less frequently relied upon. When it is used, the broker will continue to work as expected.

One key question is how the producer will interoperate with older brokers which do not support the new version of the `InitProducerId` request. For idempotent producers, we can safely bump the epoch without broker intervention, but there is no way to do do so for transactional producers. We propose in this case to immediately fail pending requests and enter the ABORTABLE_ERROR state. After the transaction is aborted, we will reset the sequence number to 0 and continue. So the only difference is that we skip the epoch bump.

Rejected Alternatives

- We considered fixing this problem in streams by being less aggressive with record deletion for repartition topics. This might make the problem less likely, but it does not fix it and we would like to have a general solution for all EOS users.
- When the broker has no state for a given producerId, it will only accept new messages as long as they begin with sequence=0. There is no guarantee that such messages aren't duplicates which were previously removed from the log or that the producer id hasn't been fenced. The initial version of this KIP attempted to add some additional validation in such cases to prevent these edge cases. After some discussion, we felt the proposed fix still had some holes and other changes in this KIP made this sufficiently unlikely in any case. This will be reconsidered in a future KIP.