KIP-890: Transactions Server-Side Defense

Status

Current state: Approved

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

We have seen hanging transactions in Kafka where the last stable offset (LSO) does not update, we can't clean the log (if the topic is compacted), and read_committed consumers get stuck.

This can happen when a message gets stuck or delayed due to networking issues or a network partition, the transaction aborts, and then the delayed message finally comes in. The delayed message case can also violate EOS if the delayed message comes in after the next addPartitionsToTxn request comes in. Effectively we may see a message from a previous (aborted) transaction become part of the next transaction. A similar event can occur if a delayed EndTxn marker comes through an a new transaction with the same epoch has started when the request is received. This can result in incorrectly completing a transaction.

Another way hanging transactions can occur is that a client is buggy and may somehow try to write to a partition before it adds the partition to the transaction. In both of these cases, we want the server to have some control to prevent these incorrect records from being written and either causing hanging transactions or violating Exactly once semantics (EOS) by including records in the wrong transaction.

The best way to avoid this issue is to:

1. **Uniquely identify transactions by bumping the producer epoch after every commit/abort marker. That way, each transaction can be identified by (producer id, epoch).**
2. **Remove the addPartitionsToTxn call and implicitly just add partitions to the transaction on the first produce request during a transaction**.

We avoid the late arrival case because the transaction is uniquely identified and fenced AND we avoid the buggy client case because we remove the need for the client to explicitly add partitions to begin the transaction.

Of course, 1 and 2 require client-side changes, so for older clients, those approaches won't apply.

3. **To cover older clients, we will ensure a transaction is ongoing before we write to a transaction. We can do this by querying the transaction coordinator and caching the result.**

Of course, for older clients, we may include a record from the previous transaction if the new transaction has started and we receive a late message. This issue can only be avoided with the new clients through the epoch bump.

Public Interfaces

We will bump the ProduceRequest/Response and TxnOffsetCommitRequest/Response version to indicate the client is using the new protocol that doesn't require adding partitions to transactions and will implicitly do so. The bump will also support new errors `ABORTABLE_ERROR`

`ABORTABLE_ERROR` can be returned on any failure server-side so that the server can indicate to the client that the transaction should be aborted. See more details in **Return Error for Non-Zero Sequence on New Producers** below.

Additionally, we will update the `END_TXN_RESPONSE` to include a bumped epoch for clients to use. We will also bump the request/response version.

```
{
  "apiKey": 26,
  "type": "response",
  "name": "EndTxnResponse",
  // Starting in version 1, on quota violation, brokers send out responses before throttling.
  //
  // Version 2 adds the support for new error code PRODUCER_FENCED.
  //
  // Version 3 enables flexible versions.
  //
  // Version 4 returns the producer epoch and producer ID.
  "validVersions": "0-4",
  "flexibleVersions": "3+",
  "fields": [
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "0+",
      "about": "The duration in milliseconds for which the request was throttled due to a quota violation, or
zero if the request did not violate any quota." },
    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The error code, or 0 if there was no error." },
    { "name": "ProducerId", "type": "int64", "versions": "4+", "entityType": "producerId",
      "about": "The producer ID." },
    { "name": "ProducerEpoch", "type": "int16", "versions": "4+",
      "about": "The current epoch associated with the producer." },
    ]
}
```

We will bump the versioning for AddPartitionsToTxn to add support for "verifyOnly" mode used for old clients so that the server can verify the partitions are in a given transaction. We will also add support to batch multiple waiting requests by transactional ID. This newer version of the request will be used server-side only and require CLUSTER authorization so clients will not be able to use this version.

AddPartitionsToTxnRequest

```
{
  "apiKey": 24,
  "type": "request",
  "listeners": ["zkBroker", "broker"],
  "name": "AddPartitionsToTxnRequest",
  // Version 1 is the same as version 0.
  //
  // Version 2 adds the support for new error code PRODUCER_FENCED.
  //
  // Version 3 enables flexible versions.
  //
  // Version 4 adds VerifyOnly field to check if partitions are already in transaction and adds support to
batch multiple transactions.
  "validVersions": "0-4",
  "flexibleVersions": "3+",
  "fields": [
    { "name": "Transactions", "type": "[]AddPartitionsToTxnTransaction", "versions":  "4+",
      "about": "List of transactions to add partitions to.", "fields": [
      { "name": "TransactionalId", "type": "string", "versions": "4+", "mapKey": true, "entityType":
"transactionalId",
        "about": "The transactional id corresponding to the transaction." },
      { "name": "ProducerId", "type": "int64", "versions": "4+", "entityType": "producerId",
        "about": "Current producer id in use by the transactional id." },
      { "name": "ProducerEpoch", "type": "int16", "versions": "4+",
        "about": "Current epoch associated with the producer id." },
      { "name": "VerifyOnly", "type": "bool", "versions": "4+", "default": false,
        "about": "Boolean to signify if we want to check if the partition is in the transaction rather than add
it." },
      { "name": "Topics", "type": "[]AddPartitionsToTxnTopic", "versions": "4+",
        "about": "The partitions to add to the transaction." }
    ]},
    { "name": "V3AndBelowTransactionalId", "type": "string", "versions": "0-3", "entityType": "transactionalId",
      "about": "The transactional id corresponding to the transaction." },
    { "name": "V3AndBelowProducerId", "type": "int64", "versions": "0-3", "entityType": "producerId",
      "about": "Current producer id in use by the transactional id." },
    { "name": "V3AndBelowProducerEpoch", "type": "int16", "versions": "0-3",
      "about": "Current epoch associated with the producer id." },
    { "name": "V3AndBelowTopics", "type": "[]AddPartitionsToTxnTopic", "versions": "0-3",
      "about": "The partitions to add to the transaction." }
  ],
  "commonStructs": [
    { "name": "AddPartitionsToTxnTopic", "versions": "0+", "fields": [
      { "name": "Name", "type": "string", "versions": "0+", "mapKey": true, "entityType": "topicName",
        "about": "The name of the topic." },
      { "name": "Partitions", "type": "[]int32", "versions": "0+",
        "about": "The partition indexes to add to the transaction" }
    ]}
  ]
}
```

AddPartitionsToTxnResponse

```
{
  "apiKey": 24,
  "type": "response",
  "name": "AddPartitionsToTxnResponse",
  // Starting in version 1, on quota violation brokers send out responses before throttling.
  //
  // Version 2 adds the support for new error code PRODUCER_FENCED.
  //
  // Version 3 enables flexible versions.
  //
  // Version 4 adds support to batch multiple transactions and a top level error code.
  "validVersions": "0-4",
  "flexibleVersions": "3+",
  "fields": [
    { "name": "ThrottleTimeMs", "type": "int32", "versions": "0+",
      "about": "Duration in milliseconds for which the request was throttled due to a quota violation, or zero
if the request did not violate any quota." },
    { "name": "ErrorCode", "type": "int16", "versions": "4+",
      "about": "The response top level error code." },
    { "name": "ResultsByTransaction", "type": "[]AddPartitionsToTxnResult", "versions": "4+",
      "about": "Results categorized by transactional ID.", "fields": [
      { "name": "TransactionalId", "type": "string", "versions": "4+", "mapKey": true, "entityType":
"transactionalId",
        "about": "The transactional id corresponding to the transaction." },
      { "name": "TopicResults", "type": "[]AddPartitionsToTxnTopicResult", "versions": "4+",
        "about": "The results for each topic." }
    ]},
    { "name": "ResultsByTopicV3AndBelow", "type": "[]AddPartitionsToTxnTopicResult", "versions": "0-3",
      "about": "The results for each topic." }
  ],
  "commonStructs": [
    { "name": "AddPartitionsToTxnTopicResult", "versions": "0+", "fields": [
      { "name": "Name", "type": "string", "versions": "0+", "mapKey": true, "entityType": "topicName",
        "about": "The topic name." },
      { "name": "ResultsByPartition", "type": "[]AddPartitionsToTxnPartitionResult", "versions": "0+",
        "about": "The results for each partition" }
    ]},
    { "name": "AddPartitionsToTxnPartitionResult", "versions": "0+", "fields": [
      { "name": "PartitionIndex", "type": "int32", "versions": "0+", "mapKey": true,
        "about": "The partition indexes." },
      { "name": "PartitionErrorCode", "type": "int16", "versions": "0+",
        "about": "The response error code." }
    ]}
  ]
}
```

```
{
  "type": "data",
  "name": "TransactionLogValue",
  // Version 1 is the first flexible version.
  // KIP-915: bumping the version will no longer make this record backward compatible.
  // We suggest to add/remove only tagged fields to maintain backward compatibility.
  "validVersions": "0-1",
  "flexibleVersions": "1+",
  "fields": [
    { "name": "ProducerId", "type": "int64", "versions": "0+",
      "about": "Producer id in use by the transactional id"},
    { "name": "ProducerEpoch", "type": "int16", "versions": "0+",
      "about": "Epoch associated with the producer id"},
    { "name": "PrevProducerId", "type": "int64", "default": -1, "taggedVersions": "1+", "tag": 0,
      "about": "Producer id in use by client when committing the transaction"},
    { "name": "NextProducerId", "type": "int64", "default": -1, "taggedVersions": "1+", "tag": 1,
      "about": "Producer id returned to the client in the epoch overflow case"},
    { "name": "TransactionTimeoutMs", "type": "int32", "versions": "0+",
      "about": "Transaction timeout in milliseconds"},
    { "name": "TransactionStatus", "type": "int8", "versions": "0+",
      "about": "TransactionState the transaction is in"},
    { "name": "TransactionPartitions", "type": "[]PartitionsSchema", "versions": "0+", "nullableVersions": "0+",
      "about": "Set of partitions involved in the transaction", "fields": [
      { "name": "Topic", "type": "string", "versions": "0+"},
      { "name": "PartitionIds", "type": "[]int32", "versions": "0+"}]},
    { "name": "TransactionLastUpdateTimestampMs", "type": "int64", "versions": "0+",
      "about": "Time the transaction was last updated"},
    { "name": "TransactionStartTimestampMs", "type": "int64", "versions": "0+",
      "about": "Time the transaction was started"}
  ]
}
```

On the client side, newer clients with the bumped produce version can read this epoch and use it for future requests.

Older clients will receive `INVALID_TXN_STATE` errors when there is not an ongoing transaction for the partition. But this error is already supported.

Configurations

We will add a new configuration to turn off the verification feature. This can be used by performance-conscious users who are concerned about the second AddPartitionsToTxn call to verify the partition and less concerned about hanging transactions and other transaction issues.

This configuration will not apply to the adding of the new partitions for newer clients since we do not expect performance differences there. For now the default will be true. If performance tests show a significant change, that can be reviewed again.

```
val TransactionPartitionVerificationEnableDoc = "Enable verification that checks that the partition has been added to the transaction before we write transactional records to the partition"
```

Metrics

VerificationTimeMs – number of milliseconds from adding partition info to the manager to the time the response is sent. This will include the round trip to the transaction coordinator if it is called. This will also account for verifications that fail before the coordinator is called.

VerificationFailureRate – rate of verifications that returned in failure either from the AddPartitionsToTxn response or through errors in the manager.

Proposed Changes

Transaction Feature Version (0)

Similar to Metadata Version, we will introduce a transaction version using the features component introduced by KIP-584: Versioning scheme for features.

This version will be used to gate the ability to use flexible fields in transactional state records – work that was started by KIP-915: Txn and Group Coordinator Downgrade Foundation and the ability to turn on the changes 1 and 2 below.

Transaction Version will behave similarly to Metadata Version as it will be dynamically changeable.

Bump Epoch on Each Transaction for New Clients (1)

In order to provide better guarantees around transactions, we should change semantics to ALWAYS bump the epoch upon the commit or abort of a transaction on newer clients (those with a higher produce version). This will allow us to uniquely identify a transaction with a producer ID and epoch. By being able to unique identify a transaction, we can better tell where one transaction ends and the next begins. This would cover the case where a message from a previous transaction incorrectly gets added to a new one and the hanging transaction case.

As we do now, we will ensure that any produce requests are using the correct epoch. Messages from previous transactions will be fenced because they will have an older epoch.

Upon the end of the transaction, the client sends the `EndTxnRequest`. When the transaction coordinator receives this, it will write the prepare commit message with a bumped epoch and send `WriteTxnMarkerRequest`s with the bumped epoch. We write the bumped epoch for compatibility so if we downgrade or switch coordinators to an older version, the markers will still be written correctly. Finally, it will send the `EndTxnResponse` with the bumped epoch (and producer ID if we overflow the epoch) to the client. Newer clients will read this epoch and set it as their own -- using it for the next transaction.

For more detail – consider the epoch bump cases for the transactional state records and EndTxn responses where we do and do not see epoch overflow:

Say we have producer ID x and epoch y. When we overflow epoch y we get producer ID z.

PREPARE
producerId: x
*previous/lastProducerId (tagged field): x
nextProducerId (tagged field): empty or z if y will overflow
producerEpoch: y + 1

- Non-overflow: Return epoch y + 1 and producer ID x
- Overflow: Return epoch 0 and producer ID z for overflow in EndTxnResponse
- Keep the previous producer ID field as x. We use this field to signify that a new client + new server (with epoch bump) set the field.
- If we retry and see epoch - 1 + and producer ID x in last seen fields and are issuing the same command (ie commit not abort) we can return (with the new epoch)

COMPLETE
producerId: x or z if y overflowed
*previous/lastProducerId (tagged field): x
nextProducerId (tagged field): empty
producerEpoch: y + 1 or 0 if we overflowed

- Non-overflow: Set the producer ID to x
- Overflow: Set producer ID to z (from nextProducerId field), set epoch to 0, and nextProduceId can go back to empty.
- Keep the previous producer ID field to x.
- If we retry and see epoch max - 1 + ID in last seen fields and are issuing the same command we can return (with current producer ID and epoch)

Once we move past the Prepare and Complete states, we don't need to worry about tagged fields and clear them, just handle state transitions as normal.

In KIP-890 part 2, when writing to a partition we will check the epoch when we add the partition, and use the existing fencing logic at the log layer (and we do not worry about the verification mechanism described in part 3).

*NOTE: When loading from the log and previous/lastProducerId field is present we can populate the lastProducerId and lastProducerEpoch fields.

Return Error for Non-Zero Sequence on New Producers

For new clients, we can once again return an error for sequences that are non-zero when there is no producer state present on the server. This will indicate we missed the 0 sequence and we don't yet want to write to the log. Previously this error was `UNKNOWN_PRODUCER_ID` but historically, this error code was handled in a complicated way. Now this scenario can be covered with the retriable OutOfOrderSequence error.

Implicitly Add Partitions to Transactions on Produce for New Clients (2)

Each broker contains cached entries about producer state that contain a field for the first offset of a transaction `currentTxnFirstOffset`. When we send a produce request with a newer version, we can check if we have a transaction `Ongoing` by checking if this field is populated. If it is there, continue handling the produce request.

If there is not any state in the broker for the partition, this is the first time producing to the partition for this transaction and we need to add it to the transaction implicitly. We will put the produce request in purgatory and send a request to the transaction coordinator to add the partition to the transaction.

We will wait for the response before continuing the produce request in case we need to abort and need to know which partitions – we don't want to write to it before we store in the transaction manager. Upon returning from the transaction coordinator, we can set the transaction as ongoing on the leader by populating `currentTxnFirstOffset` through the typical produce request handling.

This field will serve as a marker that the partition was added to the transaction and will persist across leadership changes since the information is persisted to disk. When the transaction markers are written to abort/commit the transaction this offset field is cleared.

We can also make this call for the __consumer_offsets topic when the producer's `sendOffsetsToTransaction` call is sent. When the producer contacts the group coordinator via `TxnOffsetCommitRequest` the coordinator broker can instead make the `AddPartitionsToTxnRequest` call to the transaction coordinator to add that offset partition. This means that `AddOffsetCommitsToTxnRequest` is deprecated for new clients.

New clients will have a bumped produce version and receive a valid Transaction Version supporting the feature that indicates they do not have to send AddPartitionsToTxn requests. Before this, the client will need to continue to send AddPartitionsToTxn and AddOffsetCommitsToTxn requests. Clients should continue using the older version of request (v3).

Ensure Ongoing Transaction for Older Clients (3)

As mentioned above, the broker contains information about whether a transaction is ongoing through `currentTxnFirstOffset`. When we send a produce older version produce request, we check if there is any state for the transaction on the broker. If it is there, we continue writing the produce request.

If there is not any state in the broker for the partition, we need to verify if a transactions is ongoing. We will put the produce request in purgatory and verify through the `AddPartitionsToTxnRequest` to the transaction coordinator using the `verifyOnly` flag. In the verify mode of the request, we check if the transaction is already added to the transaction coordinator for the given partition. If we do not already see the transaction during verify mode we can return `INVALID_TXN_STATE` for that partition. If the partition and transaction are there, we can return with no error.

We need to make sure the transaction is `Ongoing` and the partition is part of the transaction. If not and we returned `INVALID_TXN_STATE` from the addPartitions call, return that error in the produce response. Upon verifying this, we can continue handling the produce request that puts `currentTxnFirstOffset` in the cache. In memory state can also be stored to indicate the partition has been verified. We will check this in memory state before the record is written to avoid any race where an control marker is written after sending the verify response but before the record is written.

All future produce requests for this transaction can be verified without the request to the transaction coordinator. The `currentTxnFirstOffset` will serve as a marker that the validation completed and will persist across leadership changes since the information is persisted to disk. When the transaction markers are written to abort/commit the transaction this offset field is cleared.

Race Condition

One thing to consider here is the concurrency and potential race conditions this solution may see. One potential issue is for an abort marker to be inflight and/or written in between when the verification response comes in and the produce request written to the log. On the `WriteTxnMarkerRequest`, we update the state while the log is locked and the marker is written. Then, in the callback when we try to handle the produce in the log, we can check the state one more time to make sure no marker came in (and ended the transaction) before we wrote to it.

Compatibility, Deprecation, and Migration Plan

New Clients

All new clients will use the new produce version (with compatible brokers) and will see the benefits of epoch bumps and implicit adding to the partition for each transaction.

New clients also need to be updated to remove `addPartitionsToTxns` in order to reap the performance benefit.

New clients will also take advantage of clearly defined retriable and abortable errors. These changes will apply to both Produce and TxnOffsetCommit requests

New clients will know that brokers support the new version with the newly created transaction feature.

Old Clients

Produce Requests

For the current Java client, we would return the already existing `INVALID_TXN_STATE` error with a message indicating the transaction was not ongoing. This error will result in the batch failing, aborting the transaction and the sequence number being adjusted as to not cause issues with out of order sequence. Non-Java clients should also have this handling, but if not a comparable approach can be used. Some clients treat this error as fatal, and that is preferable to writing the state. In many cases, the producer issuing this request is a zombie and abortable/fatal distinction should not matter.

We also have a few more error codes that will be returned due to the interaction with the coordinator. Those are `INVALID_PID_MAPPING` , `CONCURRENT_TRANSACTIONS` , `COORDINATOR_LOAD_IN_PROGRESS` , `COORDINATOR_NOT_AVAILABLE` , and `NOT_COORDINATOR`

The first error is abortable on the Java client and fatal on librdkafka and perhaps other clients. This should be ok since in most cases, the request is from a zombie or misconfigured client. The others are trickier since the Java client treats all retriable errors as retriable, but not all clients do. In order to avoid fatal errors as a result of coordinator changes and concurrent transactions, we opt to use `NOT_ENOUGH_REPLICAS` with a specific error message for debugging. There are trade-offs for old client cases since we can't do the optimal choice which is to have a error that means to retry and an error that means to abort.

TxnOffsetCommit Requests

There are slightly different semantics for TxnOffsetCommits and how they handle errors. `INVALID_TXN_STATE` and `INVALID_PID_MAPPING` are fatal for this request. These will be fatal. All the retriable errors will be converted to `COORDINATOR_NOT_AVAILABLE` since `NOT_ENOUGH_REPLICAS` is not handled. These will not contain the unique error message since the api does not support that.

New Servers

New servers will indicate their ability to support the new protocol with a new transaction feature version. This is similar to metadata version, but will be used for transactional features only. The transaction feature version will be used to signal 1) that flexible fields can be written in the transactional state topic and 2) the new protocol (epoch bumps and add partitions optimization) can be used

Consider a few scenarios in the course of an upgrade – at the beginning of a transaction, the client will determine whether it should use the old or new protocol based on the client itself and latest epoch TV it sees from the brokers it is connected to.

- Any case with old client (new or old image/TV on brokers) old protocol (client doesn't support)
- New client, old image and TV on brokers old protocol (server TV is too low)
- New client, new image but not new MV on brokers old protocol (server TV is too low)
- New client, new image + TV on at least one broker new protocol (client supports and data broker has new TV – even if not all brokers have TV, it has the code to support)

Before requests, we also will check the latest epoch TV. If we see a new one, we will wait for the transaction to complete before upgrading. If we see a lower MV, we should abort the transaction and start again with the old protocol.

Old Servers

Attempting to send verifications to older brokers will be a no-op, we can just append as normal.

Downgrading a broker or switching to a broker without the new code for handling the new records should gracefully ignore the tagged fields and write the commit markers with the bumped epoch. This should not affect old brokers or consumers.

Attempting to use a new client with old server code that doesn't add partitions will result in the new client falling back to old behavior. It is possible that some brokers will have old code and some will have new code. In this case we can not ignore the add partitions calls for new clients. In this case, we should fail with an abortable error and signal to the client the old protocol should be used as described above

Test Plan

Unit/Integration testing will be done to test the various hanging transaction scenarios described above. Tests will also be done to ensure client compatibility between various image and transaction feature versions.

Rejected Alternatives

Use Metadata Version to gate features

Introducing a separate feature gives greater control over the rollout process and allows us greater control over turning the feature off. We don't foresee a reason to require us turning off/downgrading, but it feels better to have the option and the ability to not affect/block other features when making any changes. Creating a new feature paves the way for other areas to create their own features (ie, Group Coordinator for KIP-848) This usage is consistent with the original intent of KIP-584.

In the overflow case, have a CompleteCommit record with the old producer ID + the new producer ID written in a pseudo InitProducerIdRecord

This is closer to how the old client works in the epoch overflow case. We could also atomically commit both records. However, since the records are cleaned up via compaction with the transactional ID as the key, we can lose information about relationships between producer IDs.

Although the complete record with new producer ID is a bit strange seeming, it is best for when the server downgrades and the client has the new producer ID. It also doesn't have any problems with correctness.

Bump the epoch on EndTxn but write the marker with the old epoch

Pre KIP-890 part 2 old producers are fenced in the timeout case by bumping the epoch and writing abort markers.

This isn't needed in KIP-890 part 2 since a marker indicates the end of the transaction and we expect no more records for a given epoch once the marker is written.

However, given the behavior we already have for the timeout and fencing, it will be more compatible with older versions if we bump the epoch in the end marker for every transaction.

In KIP-890 part 2, when writing to a partition we will check the epoch when we add the partition, and use the existing fencing logic at the log layer (and we do not worry about the verification mechanism)

Return Abortable Error for TxnOffsetCommitRequests

Instead of `INVALID_TXN_STATE` and `INVALID_PID_MAPPING` we considered using `UNKNOWN_MEMBER_ID` which is abortable. However, this is not a clear message and is not guaranteed to be abortable on non-Java clients. Since we can't specify a message in the response, we thought it would be better to just send the actual (but fatal) errors.

Use a more specific error that works for older Java clients but may be fatal on other older clients

KIP-890 Part 1 tries to address hanging transactions on old clients. Thus, the produce version can not be bumped and no new errors can be added. The older java client has a notion of retriable and abortable errors -- retriable errors are defined as such by extending the retriable error class, fatal errors are defined explicitly, and abortable errors are the remaining. However, many other clients treat non specified errors as fatal and that means many retriable errors kill the application. Stuck between having specific errors for Java clients that are handled correctly (ie we retry) or specific fatal errors for cases that should not be fatal, we opted for a middle ground of non-specific error, but a message in the response to specify.

InvalidRecord Error Returned When No Ongoing Transactions for Older Clients

`INVALID_RECORD` was a proposed error to send when the partition was not part of the transaction since it was already returned by the producer and treated as an abortable error. This seemed like a good choice for older clients. However, this error has not existed as long as transactions have existed, so some clients may not parse this error. Therefore, `INVALID_TXN_STATE` which is also an abortable error should be used instead.

Overhaul Transactions Protocol + Begin Transactions

Part of the reason why hanging transactions are an issue is that the client must explicitly add partitions to a transaction. As part of removing the AddPartitionsToTxn request, there was some consideration to redoing the protocol from scratch and introducing a unique transactional ID per transaction. This could remove the reliance on the producer ID + epoch.

1. BeginTxn – send to the txn coordinator to signal the beginning of a transaction, producer is given a unique transactional id
2. Produce – we send produce requests with the given transactional ID to the leaders and implicitly add them to the transaction.
3. EndTxn – send to the txn coordinator to signal the end of the transaction – commit markers are written to all the leaders, transactional id is fenced.

Though there may be some reason to remove reliance on producer IDs (in case we want to make changes there in the future), the immediate need was not apparent.

Begin Transactions Marker

Building off of the bumping epoch on each transaction approach, we could also try to identify unique transactions by including a begin transactions marker. This would accomplish a few goals:

1. Clearly mark in the log the start of a transaction which could aid in debugging
2. Persist the information about the start of a transaction locally – we wouldn't have to send the data to new leaders since it would be contained in the log.
3. Supplement the epoch bump approach (or potentially implement without epoch bumps) so that the transaction coordinator and leader can guard against hanging transactions/records from a previous transaction being added to a new one.
4. Using the existing WriteTxnMarkerRequest pipeline means this won't be a huge overhaul – though we may want to make the call synchronous to avoid the "concurrent transactions" issue we see when writing the end marker (ie, some time to propagate that requires retries – the same issues we saw with the coordinator leader path)

There are some cons to this approach which led us to ultimately decided not to take this path, at least in v1:

1. It requires changing the record format, which may not be too difficult
2. We still need to send the transaction state to the leader to write the marker (and persist it to all the replicas) which takes some time
3. We could still implement this approach at a later date if we implement the proposed current solution.

Return LogStart and LogEndOffsets in WriteTxnMarkers Request

The idea is that this would provide more information for debugging. However, the information is found in ProducerStateManager and hard to propagate up to the response – especially since this request is handled as and append to the log. Any changes to those methods would also affect the produce path.

Instead, perhaps we could include a log message with this information from the broker side.

AddPartitionsToTxn Optimization for Older Clients

We could introduce an optimization to store the state on the leader on the addPartitionsToTxn request (issuing the a new api from the TxnCoordinator to the leader). This would make the first request after adding the partition faster since we may already have the state and can produce without an extra API call. However, in the case where the message is too slow, we would need to keep retrying until the information came in or we timeout (since the transaction is not ongoing and such state will never arrive). Failing fast is pretty important, so we decided against this approach.

DescribeTransactions call for Older Clients

Originally the plan was to use the DescribeTransactions all to check if a transaction was ongoing. However, the logic to send from the leader is already pretty complicated, so it will be easier to handle a single request type.