Kafka Exactly Once - Solving the problem of spurious OutOfOrderSequence errors

Background

In the discussion of KIP-185: Make exactly once in order delivery per partition the default producer setting, the following point regarding the `OutOfOrderSequenceException` was raised:

- The `OutOfOrderSequenceException` indicates that there has been data loss on the broker.. ie. a previously acknowledged message no longer exists. For most part, this should only occur in rare situations (simultaneous power outages, multiple disk losses, software bugs resulting in data corruption, etc.).
- However, there is another perfectly normal scenario where data is removed: in particular, data could be deleted because it is old and crosses the retention threshold.
- Hence, if a producer remains inactive for longer than a topic's retention period, we could get an `OutOfOrderSequence` which is a false positive: the data is removed through valid processes, and this isn't an error.
- In the current implementation of the code, we currently raise an `OutOfOrderSequenceException` when we get a duplicate of a batch which is not at the tail of the log. This is also confusing, and a more clear error would be the `DuplicateSequenceException` in this case.

We would like to eliminate the possibility of getting spurious `OutOfOrderSequenceExceptions` – when you get it, it should always mean data loss and should be taken very seriously.

Design

Essentially, we want to distinguish between the case where a producer's state is removed from the broker because the retention time has elapsed, and when the state is lost due to some problem in the system.

One solution is described here:

1. When the producer metadata is removed from the `ProducerStateManager` on the broker due to retention, the next `ProduceRequest` from the client will arrive with the existing producer id and with a non-zero sequence. Currently this results in an `OutOfOrderSequenceException` returned by the broker, since the broker can't find any metadata and gets a non-zero sequence. This isn't strictly correct, and we propose introducing a new `UnknownProducerException` and returning this instead.
2. The client can treat the `UnknownProducerException` as a non-fatal error and just reinitialize the producer and continue on its merry way *in most cases*.
3. However, the above solution opens a hole: if the first write from the producer is actually lost (maybe due to a simultaneous power outage, multiple disk failures, etc.), we would not detect it. In particular, the first write with sequence = 0 is written, but then the records are lost on the broker. The next write with sequence=N would get an `UknownProducerException` and with the protocol above would simply be retried. Hence the fact that a message was lost would never be raised to the application. This applies to the first write because it is only at the front of the log where there could be a confusion between removal due to retention or loss due to an unforeseen circumstance.
4. We can solve the situation in (3), by keeping track of the last ack'd offset on the producer, and also returning the log start offset in each `ProduceResponse`. With these two pieces of information, we can be sure that an `UknownProducerException` is valid if the log start offset returned along with the error code is greater than the last ack'd offset. This means that the front of the log has been truncated, causing the producer to become unknown. In this case, there is no unwanted data loss and the last batch can simply be retried. If we get an `UnkownProducerExceptior` but the log start offset is *not* greater than the last ack'd offset, then the record has been not been lost due to the retention period elapsing, and this should be treated as a fatal error.
5. If we a trying to append a batch with a sequence less than the sequence of the batch at the tail of the log, we should return a `DuplicateSequenceException` instead of an `OutOfOrderSequencException`
6. With the changes above, an `OutOfOrderSequenceException` would always mean real data loss. An `UnkownProducerException` may mean data loss, and the information of the last ack'd offset and log start offset will enable us to disambiguate.

Level of Effort

1. Client side changes to track the last ack'd offset and correctly interpret an `UnknownProducerException` and either retry it or raise it as an error – 1 day.
2. Broker side changes to raise the `UnkownProducerException`– 0.25 days.
3. Updates to the protocol to return the `logStartOffset` per partition (with KIP) - 2 days.
4. System tests + Debugging - 2 days

Total : 1.25 weeks.