KIP-207: Offsets returned by ListOffsetsResponse should be monotonically increasing even during a partition leader change

Status

Current state: *Accepted*

Discussion thread: *https://www.mail-archive.com/dev@kafka.apache.org/msg81074.html*

JIRA: *KAFKA-2334*

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

Motivation

ListOffsetsRequest returns the latest offset for each partition. These offsets should advance monotonically as new messages are added to the partition. However, sometimes, right after a partition leadership change, the message offsets returned by *ListOffsetsRequest* can actually go backwards.

Because it happens very rarely, applications that use Kafka are usually not prepared for non-monotonic offset behavior. Applications, such as connectors for Spark Streaming, may crash or misbehave. To avoid these issues, we should fix this corner case so that offsets advance monotonically even after a leader election.

Proposed Changes

After a successful partition leadership election, a former follower is now the leader. However, the high water mark on the former follower may be behind the high water mark on the old, failed leader. This is the cause of the non-monotonic behavior immediately after the election.

What we would like to do is wait until the new leader's high water mark has caught up with the messages already in its log. To implement this, during the transition from follower to leader, the broker can record the current *logEnd* for the partition. Then, it can refuse to answer *ListOffsetsRequest* for that partition until the high water mark has caught up with this value.

The period when the offset is unavailable should be brief. During this period, the broker should simply return a retriable exception when it is asked for the offset of the partition. For current versions of ListOffsetsRequest, this exception can be *LeaderNotAvailableException.* For new versions of ListOffsetsRequest, we can return a new, more precise exception. The main advantage of creating a new exception is that the client knows it can avoid re-fetching metadata. A second advantage is that the more precise error message may help with debugging on the client side. During this brief time period, we will be able to fetch records from the new leader, but not find the latest offset for the partition.

This behavior will apply only when answering requests from clients. Therefore, there will be no impact on *ReplicaFetcherThread,* or other parts of the broker that make *ListOffsetsRequests* to other brokers. We can distinguish broker requests from client requests by looking at the *replicaId* field of the RPC. (Hence, other brokers can still use ListOffsetsRequests to figure out which replicas has the longest log for a partition.)

When unclean leader elections are enabled, data loss is possible. So we cannot guarantee that offsets will always go forwards, even in theory, in this mode. Therefore, when unclean leader elections are enabled on the broker, the KIP-207 behavior will not apply.

Public Interfaces

There will be a new version of *ListOffsetsResponse* API. This will be the same as the existing one, except that we can return a new exception, *OffsetNotAvailableException*, for a partition. This new exception will be a subclass of *RetriableException*.

The KIP-207 behavior applies to all *ListOffsetsRequests*, whether they are for the latest offset, the earliest offset, or a time-based offset. Since leader changes are rare, the performance impact should be very small. Treating all *ListOffsetsRequests* the same simplifies the code. This also avoids creating awkward situations where we can locate the offset for a certain time T, and then cannot locate the offset for that time following a leader change.

Compatibility

The new *OffsetNotAvailableException* mentioned earlier will be sent only to post-KIP-207 clients. As mentioned earlier, pre-KIP-207 clients will receive *LeaderNotAvailableException*. Therefore, older clients should be able to communicate with servers implementing KIP-207. Similarly, because the client changes are limited to handling a single additional exception, post-KIP-207 clients can communicate with pre-KIP-207 brokers.

Rejected Alternatives

Rather than returning a retriable exception, the broker could simply put the *ListOffsetsRequest* into a purgatory structure until the offset was available. This avoids the need for the client to poll the server. We would create a new version of the ListOffsetsRequest RPC which adds a maximum timeout field.

However, adding a new purgatory structure would increase the complexity of the code substantially. Since this case is a corner case which only happens for a few seconds after a leader election, the extra performance does not seem worth it. It is also awkward to put *ListOffsetsRequest* into a purgatory, because each request could ask about multiple partitions. The results for some partitions might be blocked because other partitions were not ready to return their offsets. Finally, because an RPC revisions is needed, this approach would solve the problem for new clients, but not for older ones. The approach above avoids these problems: it avoids adding a new purgatory structure, allows us to give back results immediately for the partitions where we know those results, and solves the problem for older clients as well as newer ones.