KIP-58 - Make Log Compaction Point Configurable

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *KAFKA-1981*

Released: 0.10.1.0

Motivation

Currently Kafka's log compaction gives minimal control over what portion of the log remains uncompacted. There is a setting that prevents compaction until a certain dirty ratio has been reached but this does not provide any upper bound on how much of the log's head will remain uncompacted once it runs. Although the segment currently being written will never be compacted, this could leave as little as one message uncompacted.

As a result, it is impossible to be guaranteed that a consumer will get every update to a compacted topic. Even if the consumer falls behind by a single message it might get the compacted version.

One particularly relevant use case is database state replication through change data capture. This use case is specifically called out in the Kafka documentation for the compaction feature under "Database change subscription". It is convenient to produce this data in multiple topics (e.g. one per source table) and/or partitions. However, in order to be able to recreate a database state at a point of transactional consistency some coordination across topics/partions is required (e.g. a separate _checkpoint_ topic with the offsets at each transaction commit). If the table topics are all independently compacting there is currently no way to be assured that any given checkpoint can be materialized as the checkpointed offset for any given topic may have been compacted such that some keys may be taking on some subsequently inserted values. (Details: https://gist.github.com/ewasserman/f8c892c2e7a9cf26ee46)

Another use case is to handle application mistakes. For example, a compacted topic could be the source of truth for certain type of data. If there is an application error, some incorrect data may be published to the topic. When compaction is triggered, those incorrect data can wipe out the last correct message associated with a key. Being able to delay the compaction based on a configurable amount of time will allow a user to preserve those last known correct messages after an application error is discovered but before compaction destroys the correct data.

Public Interfaces

This proposal includes new configurations for controlling compaction. The log cleaner can be configured retain a minimum amount of the uncompacted head of the log. This is enabled by setting the compaction lag:

```
log.cleaner.min.compaction.lag.ms
```

for setting minimum message age in milliseconds. This has a similar per-topic configuration:

```
min.compaction.lag.ms
```

This lag configuration defaults to zero so that if not set, all log segments are eligible for compaction except for the last segment (i.e. the one currently being written). The active segment will not be compacted even if all of the compaction lag constraint is satisfied. This leaves unchanged the current behavior. If the compaction lag is greater than zero then compaction of the segments of the logs containing any messages that do not satisfy the lag constraint will not be compacted. In particular this allows for the example use case like: "any consumer that is no more than 1 hour behind will get every message."

Proposed Changes

Introduce an additional configuration to topics that guarantee a minimum portion of the head of the log will remain uncompacted. That is, offer a guarantee that a consumer that does not lag too far behind will get every update to a compacted topic. This can be used to set constraints on the minimum _distance_ from the topic head that will remain uncompacted, where distance is defined in terms of time since insertion (i.e. message age).

The basic behavior of the compaction ratio to trigger and prioritize compaction order will not be altered. However, the ratio's definition will be expanded to become the ratio of "compactable" to compactable plus compacted message sizes. Where ***compactable*** includes log segments that are neither the active segment nor those prohibited from being compacted because they contain messages that do not satisfy the new lag constraint.

The time lag guarantee can be satisfied by preventing compaction of any segment containing a message or messages within the time lag. KIP-33 - Add a time based log index provides a mechanism for this to be accurately computed for a log segment.

Compatibility, Deprecation, and Migration Plan

The proposed change is backward compatible.

Rejected Alternatives

The database replication use case could be satisfied using a combination of "snapshot" and "journal" topics for each table. The journal topics could use regular time-based deletion. There would need to be some external process periodically creating new snapshots from the most recent snapshots and the journals.

While it would be straightforward to introduce other types of "lag" (e.g. aggregate message size, or message count) there were not sufficient motivating use cases to justify their inclusion at this time.