KIP-71: Enable log compaction and deletion to co-exist

- Status
- Motivation
- Rejected Alternatives

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *KAFKA-4015*

Released: 0.10.1.0

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

Motivation

For some usages, i.e., join windows in Kafka Streams, it is desirable to have logs that are both compacted and deleted. In these types of applications you may have windows of time with many versions of key, during the window you only want to retain the latest version of the key, however once the window has expired you would like to have the segments for the window deleted. With both compact and delete enabled retention.ms of the changelog would be set to a value greater than the retention of the window. Although old windows wont automatically be removed on expiration they will eventually be removed by the broker as the old segments expire. Kafka doesn't currently support these semantics as compaction and deletion are exclusive.

Enabling this will also be useful in other scenarios, i.e., any ingest of data where you only care about the latest value for a particular key, but disk constraints mean you can't keep the entire keyset.

Public Interfaces

Modify cleanup.policy to take a comma separated list of valid policies, i.e., cleanup.policy=compact,delete

Proposed Changes

Modify cleanup.policy to take a comma separated list of valid policies. When cleanup.policy=compact,delete is set, both compact and delete cleanup strategies will run.

Implementation outline

The LogCleaner.CleanerThread is currently responsible for triggering the cleaning of topics with cleanup.policy=delete. We will extend this to also support cleanup.policy=compact_and_delete. In the cleanOrSleep method we'd first run compaction and then run deletion. We'd need to add some extra code to Log to check if we have segments ready to be deleted and add any that are ready to the inProgress map (so we don't get multiple threads trying to delete the same segments), run the delete operation, and then remove them from the inProgress map (this is the same as it currently works for compacted logs).

There is no change for topics with cleanup.policy=delete, i.e, the cleanup will still be scheduled via LogManager. The benefits of this approach are that it requires no further locking, all compacted topic cleaning is triggered from LogCleaner.CleanerThread and topics that are cleanup.policy=delete are not impacted.

Compatibility, Deprecation, and Migration Plan

- *No impact on existing users*

Rejected Alternatives

Add another Lock to Log.scala. We considered adding a ReentrantLock to Log. The CleanerThread and LogManager would try and acquire this lock before attempting to clean LogSegments. We rejected this approach as there is already a fairly complex locking hierarchy in partition / replica / log / logsegment, and we'd prefer to not add another lock.

Move all cleanup code to LogCleaner and use locking approach above. This is a hybrid of our proposed solution and the rejected alternative above. Rejected due to the same reason as above.

Introduce another config log.compact and deprecate cleanup.policy. This would have been a backward compatible change requiring a migration path for existing uses. It also introduced some awkwardness around supporting the existing usage of cleanup.policy=compact, i.e., you would also need to ensure that replication.ms and replication.bytes were set to -1.