KIP-280: Enhanced log compaction

Status

Current state: Accepted (vote)

Discussion thread: https://lists.apache.org/thread.html/67fcfe37169bdbabdbecc30686ccba0f5f27e193c468a1fe5d0062ed@%3Cdev.kafka.apache.org%3E

~~**Old Discussion thread**: https://lists.apache.org/thread.html/f44317eb6cd34f91966654c80509d4a457dbbcedd02b86645782be67@%3Cdev.kafka.apache.org%3E~~

JIRA: Unable to render Jira issues macro, execution error.

PULL REQUEST: *https://github.com/apache/kafka/pull/8103*

Motivation

Current log compaction is based on the server side view i.e. compacted based on record offset and the offset is by the order when the record was received on the broker side. So for the same key, only the highest offset record is kept after compaction so that Kafka is able to reconstruct the current state of the events in a "*most recent snapshot*" approach. The issue then occurs when the insertion order is not guaranteed, which causes the log compaction to keep the wrong state. This can be easily replicated when using a multi-threaded (or simply multiple) producer(s), or when sending the events asynchronously. The following is an example:

Producer 1 tries to send a message <K1, V1> to topic A partition p1. Producer 2 tries to send a message <K1, V2> to the same (i.e. topic A partition p1). On the producer side, we clearly preserve an order for the two messages, <K1, V1> <K1, V2>. But on the server side, this order can be random, meaning, message <K1, V1> could have a higher offset due to the fact this message is received later than <K1, V2>. When compaction happens, <K1, V1> will be kept, and clearly this is not what is intended.

To resolve the above issue, we are proposing to add a feature to support compaction based on producer signal (i.e. adding 2 more compaction strategies, record timestamp and header sequence/version) and keeping the current compaction (i.e. offset based) as the default compaction for the backward compatibility. By this way, the producer will have an option to own and control the record ordering. As the log compaction is at the topic level and a broker can have multiple topics, keeping the compaction strategy configuration at topic level will be ideal. As the proposed configuration is at the topic level, the user can choose to enable a different compaction strategy for a subset of compact topics or at the broker level for all topics within the broker. While this proposal only supports two compaction strategies, it leaves the option open to add more compaction strategy in future.

Special case where we need to retain LEO (log-end-offset) record / create an empty message batch for non-offset based compaction strategy:

> *Today with the offset-only compaction strategy, the last record of the log (we call it the log-end-record, whose offset is log-end-offset) would always be preserved and not compacted. This is kinda important for replication since followers reason about the log-end-offset on the leader. Consider this case: three replicas of a partition, leader 1 and follower 2 and 3.*
>
> *Leader 1 has records a, b, c, d and d is the current last record of the partition, the current log-end-offset is 3 (assuming record a's offset is 0).*
>
> *Follower 2 has replicated a, b, c, d. Log-end-offset is 3 Follower 3 has replicated a, b, c but not yet replicated d. Log-end-offset is 2.*
>
> *NOTE: that the compaction triggering are independent on brokers, it is possible that leader 1 triggers compaction and deletes record d, while other followers have not triggered compaction yet. At this moment the leader's log becomes a, b, c. Now let's say follower 3 fetch from leader after the compaction, it will no longer see record d.*
>
> *Now suppose there's a leader migration and follower 3 becomes the new leader, it would accept new appends (say, it's e), and record e would be appended at *offset 3 *on new leader 3's log. But follower 2's offset 3's record is d still. Later let's say follower 2 also triggers compaction and also fetches the new record e from new leader 3:*
>
> *Follower 2's log would be* a(0), b(1), c(2), e(4)* where the numbers in brackets are offset number; while leader 3's log would be *a(0), b(1), c(2), e (3)*. Now you see the two logs diverges in offsets, although their log entries are the same.*

One way to resolve this, is to simply never remove the last message during compaction. Another way (suggested by Jason in the old VOTE thread) is to create an empty message batch to "take up" that offset slot.

Acknowledgement: we thank the previous author of this KIP proposal, Luís Cabral.

Public Interfaces

Adding below new configuration properties in both broker level and topic level configuration:

Broker Level:

1. *log.cleaner.compaction.strategy*
    - Broker level it is optional and default it to offset
2. *log.cleaner.compaction.strategy.header*

Topic Level:

1. *compaction.strategy*
    - default to broker level compact strategy
2. *compaction.strategy.header*

Proposed Changes

- The current behavior should remain as the default in order to minimize impact on already existing clients and avoid any migration efforts;
- New Configuration:
    1. *log.cleaner.compaction.strategy*
        - The active compaction strategy to use;
        - Accepts values "*offset*", "*timestamp*" and *"header"*, allowing for further strategies to be added in the future as needed;
    2. *log.cleaner.compaction.strategy.header*
        - Configuration sub-set to use when the strategy is set to *"header"*;

- Compaction Strategies:
    - *"offset"*
        - The current behavior is active, compacting the logs purely based on offset;
        - Also used when the configuration is either empty or not present, making this the default strategy;
    - "timestamp"
        - The record timestamp will be used to determine which record to keep, in a 'keep-highest' approach;
        - When both records being compared contain an equal timestamp, then the record with the highest offset will be kept;
        - This requires caching also the timestamp field during compaction, in addition to the base offset, so each record being compacted will suffer a memory increase from 8 bytes to 16 bytes when using this strategy.
    - *"header"*
        - Searches the record for a header key that matches the configured value on *"log.cleaner.compaction.strategy.header";*
            - *as the header can have duplicates, will pick the last occurrence of the header key*
        - If both records being compared do not have a matching header key, then the record with the highest offset will be kept;
        - If a header key that matches the configuration exists, then the header value (which must be of type *"long" - 8 bytes*) will be used to determine which record to keep, in a 'keep-highest' approach;
        - If both records being compared contain an equal header value, then the record with the highest offset will be kept;
        - If only one of the records being compared has a matching header, then this record is kept, as the other record is considered to be anomalous;
        - This requires caching also the header value during compaction, in addition to the base offset, so each record being compacted will suffer a memory increase from 8 bytes to 16 bytes when using this strategy.
- Retain LEO (log-end-offset) record during compaction

Compatibility, Deprecation, and Migration Plan

Following the above proposed changes, there are no compatibility issues. However to migrate existing topic to use header strategy, we are proposing below sequence to avoid inconsistency during migration:

- Update producer to send the header value in all record.
- Roll out the producer first to all clusters.
- Once all producers sending the header value confirmed, update the topic config on the broker side with the header strategy.
- Note:
    - any existing topic migration, the already compacted log still remains as such (i.e. offset based) and only the new logs will get compacted using the new strategy once the topic config updated with the strategy.
    - in any case if topic strategy needs roll back to default offset strategy, first topic config on the broker side should be updated to offset strategy and then producer can stop generating the header value.

Recommendations

- For scenarios like the low produce rate, the topic partition remaining ineligible for compaction for an unbounded duration where by "delete.retention.ms" triggers that removes the tombstone record if exist. In that case we recommend the Kafka users to have "segment.ms" & "max.compaction.lag.ms" (as compaction won't happen on active segment) to be smaller than the "delete.retention.ms".
- As this KIP is introducing configurable compaction strategy, the Consumer should be aware and follow the same compact strategy as in broker to avoid inconsistency on what records to keep.

Rejected Alternatives

(This section remains the same as previous proposal.)

- Producer to use Transaction with EOS to address Zombie Producer issue
  - Currently it is available only for the java clients (c++ support is in progress)
  - Adds additional overhead because of transaction to just address the compaction incorrect state issue that explained above
- Stream the data out of Kafka and perform Event Sourcing there
  - This would mean creating an in-house solution, which makes Kafka irrelevant in the design, and so its best left as a last-approach in case no solution is found on Kafka-side
- Guarantee insertion order on the producer
  - Not viable as keeping this logic synchronized greatly reduces the event throughput
- Check the version before sending the event to Kafka
  - Similar to the previous point, though it adds extra complexity as race-conditions may arise when attempting to compare
- Caching the record version as a byte array and perform the comparisons between records using a lexicographic byte array comparator
  - This adds greater flexibility on the client side, but allowing a variable byte array size to be used raises concerns about memory usage by the cache
- Always search the headers for a key matching whatever is configured, so if a header "timestamp" exists then it could be used by the compaction mechanism
  - This introduces backwards compatibility issues, as any headers are allowed without this change and the compaction is not affected at all.
  - Even if ignoring the previous point, this may cause API issues as, for example, the topic may be designed with "offset" compaction, which makes it unclear if the Producer should then provide a header "offset" or if the internal offset is meant to be used.
- Provide the configuration for the individual topics
  - None of the configurations for log compaction are available at topic level, so adding it there is not a part of this KIP