Kafka Enriched Message Metadata

Update: the scope of this proposal is narrowed to the kafka core properties with the focus on compression / log compaction only now. We leave other issues such as auditing that may involve application properties for future discussion.

Motivation

We have been discussing about several Kafka problems:

1. Log cleaning dependence on the log rolling policy (KAFKA-979): today we clean up log data at the granularity of log segments, and only considering non-active log segments. As a result, if a log does not roll for a long time, the specified log cleaning policy may not be honored. This can cause unexpected amount of data duplicates when the consumer offsets are reset to "smallest".

2. Log segment timestamp not persist during partition migration / broker restart (KAFKA-881 / KAFKA-1379): related to the previous issue (KAFKA-979), today the time-based log rolling mechanism depends on the creation time of the log segments, which will be changed when partition migrates or broker restarts, violating the log rolling policy. Requests about adding timestamps into the messages as well as the index files have also been proposed (KAFKA-1403).

3. Mirror Maker de-compress / re-compress issue (KAFKA-1001): MM need to always do decompression at the consumer side and then re-compress the messages at the producer side in case there are keyed messages; this can lead to high CPU / memory usage and also risk of data loss due to too-large-message after re-compression.

4. Broker de-compress / re-compress issue (KAFKA-527): broker needs to de-/re-compress messages just for assigning their offsets upon receiving compressed messages, leading to high CPU / memory usage.

These issues may be independently resolvable with separate solutions, but they actually come from the same root cause: some per-message metadata are either lacking (such as timestamps for log cleaning, wrapped offsets for avoiding de-/re-compression on broker / mirror maker, control messages) or being written as part of the message content, which requires Kafka to open the message content (including de-serialize / de-compress) thought it should not care about other values. Therefore, by enriching our current message metadata it is possible for us to kill them all in one stone. This page is made to inspire discussions about feasibility of this "one stone" approach.

Current Message Format

Enriching message metadata would be a wire protocol change. Fortunately that is affordable since we already add versions of message protocols in 0.8. The current request protocol can be found here. In short, a message is formatted as the following:

```
MessageAndOffset => MessageSize Offset Message
  MessageSize => int32
  Offset => int64

  Message => Crc MagicByte Attributes KeyLength Key ValueLength Value
    Crc => int32
    MagicByte => int8
    Attributes => int8
    KeyLength => int32
    Key => bytes
    ValueLength => int32
    Value => bytes
```

The magic byte (int8) contains the version id of the message, currently set to 0.

The attribute byte (int8) holds metadata attributes about the message. The lowest 2 bits contain the compression codec used for the message. The other bits are currently set to 0.

The key / value field can be omitted if the keylength / valuelength field is set to -1.

For compressed message, the offset field stores the last wrapped message's offset.

Proposed New Message Format

1. We would like to add the "enqueue" timestamp that is set by the broker upon receiving the message as the first class of the message header.
2. We would like to add some "Kafka properties" to the message metadata that are core to Kafka that brokers care about them. Examples include:

- Timestamps upon reception (for any messages).
- Number of wrapped messages (for compressed messages).
- Wrapped message set relative offsets honor-ship (for compressed messages).
- Indicator whether the wrapped message set contain keys (for compressed messages).
- ...

Here is the proposed new message format:

```
MessageAndOffset => MessageSize Offset Message
  MessageSize => int32
  Offset => int64


  Message => Crc MagicByte Attributes Timestamp KafkaTagLength [KafkaTag] KeyLength Key ValueLength Value
    Crc => int32
    MagicByte => int8
    Attributes => int8
    Timestamp => int32

    KafkaTagLength = > int32
    KafkaTag =>
      KafkaTagId => int8
      TagValue => [different types]

    KeyLength => int32
    Key => bytes
    ValueLength => int32
    Value => bytes
```

Here is a summary of the changes:

- MagicByte value is set to "1".

- For compressed message, the offset field stores the starting offset (i.e. the offset of the first wrapped message).
    - The inner compressed messages' offset field will store the relative offset against the starting offset (i.e., 0, 1, 2 ...)
    - The offset of the inner message will then be the starting offset + relative offset.
    - With log compaction turned on, the relative offsets will be non consecutive.
    - When the compressed message is replicated to other clusters, the relative offsets need to be ignored and the offset of the inner message is then calculated as the starting offset + i (if it is the i-th wrapped message)

- We will use the lowest 4 bits of the Attribute byte to indicate the type of the message:
    - normal - uncompressed
    - normal - gzip compressed
    - normal - snappy compressed
    - ...
    - ... (below are future possible types)
    - ...
    - control - leader epoch (from leader)
    - control - start transaction (from transaction coordinator)
    - control - commit transaction (from transaction coordinator)
    - control - abort transaction (from transaction coordinator)
    - ...

- Kafka tags are identified by pre-defined IDs, and hence can be stored in a compact way. For example:
    - 0: timestamp => int64, Unix timestamp set by the broker upon receipt, and hence can change while the message is going through the pipeline via MM.
    - 32: num_messages => int32, number of wrapped messages in the message set; this is only used for the compressed messages.

    - ...

- KafkaTagsLength specifies the total bytes of the tags field, which can be used iterate through the tags or skip the whole collection directly.
- Broker reading an unknown tag id will simply ignore the tag, and if there is a necessary tag that is not present it will use some default value / log exceptions. By doing this Kafka tag protocol change would not require a strict broker / client upgrade.

With the new format each message's metadata size increased by 8 bytes in the best case (KafkaTagsLength = -1 indicate empty tags).

Using the New Message Metadata

Here is a brief description about how we are going to use the new metadata to solve the above mentioned issues.

Broker Offset Reassignment (KAFKA-527)

When producer compressed the message, write the relative offset value in the raw message's offset field. Leave the wrapped message's offset blank.

When broker receives a compressed message, it only needs to set the wrapped message's offset and hence do not need to de-/re-compress message sets.

When the log cleaner is compacting log segments, when merging multiple wrapped messages into one it needs to update the raw message's relative offset values (note this will leave "holes" inside the new wrapped message).

MirrorMaker Re-factoring (KAFKA-1001)

If non of the wrapped raw messages contains key, the producer can set the non-keyed indicator of the compressed message to true; otherwise set to false.

When MM's consumers gets the compressed message, if the non-keyed indicator is set it does not need to de-compress it, otherwise it needs to compress it.

If it does not de-compress the message, reset the honor-ship flag of the relative message so that they will be treated as continuous offsets.

The new producer's API needs to be augmented to send an already compressed message (and hence not adding another message metadata into it any more).

When consumers decompress message set, it will return the message with its offset either by the "message set starting offset" + "relative offset" if the honor-ship flag is set; or "message set starting offset" + "index of the message in set" otherwise.

Log Compaction / Log Cleaning (KAFKA-881, KAFKA-979)

Add the timestamp field into the index file, which will then look like <offset, time-stamp, physical position>.

The log compaction and log cleaning method can now be incorporated in the same background thread, who will do the following upon waken up (remember a topic-partition can either be compacted or cleaned, but not both):

1. If compaction is used:
    a. Merge multiple consecutive message sets into one => just need to remember the starting offset of the first message set and changing the relative offset values.
    b. If the relative offset honor-ship is not set do not use the relative offset but just use the index of the message in the set.
2. If log cleaning is used:
    a. Loop over the segments from head to tail, checking the timestamp tag of the last message from the index file. If the timestamp is old enough, delete the whole segment.
    b. For the first segment whose last message's timestamp is still not old enough, do binary search based on index file's timestamp and do a head truncation on that segment file.

With this the log cleaning mechanism is no longer dependent on the log rolling policy.