KIP-931: Flag to ignore unused message attribute field

This page is meant as a template for writing a KIP. To create a KIP choose Tools->Copy on this page and modify with your content and replace the heading with the next KIP number and a description of your issue. Replace anything in italics with your own description.

Status

Current state: *"discarded"*

Discussion thread: *here*

JIRA: *here*

Note: discarding this KIP since the benefits of the space savings outweigh the cost incurred for message conversion. And there are also planned changes for version v.3, like:

3. v3 of the record format should make it cheaper to make changes in the future (perhaps it could support tagged fields or similar)

4. We'd want to fix other known issues at the same time (eg log append time should always be available, there may be others)

5. We should consider whether we would want to introduce a user header that is at the batch level vs record level for efficiency reasons

[Additional proposal] Changing the batch format proposal

Problem: Currently, we perform validation of data schema (such as verifying monotonically increasing offsets etc.). To do that, we have to read the headers of each message in a batch. For a compressed batch, this means that we have to decompress the entire batch including the payload (key/value) even though we don't really require any information out of them.

Solution: If we change the ordering of messages in a batch to prefix with headers of all messages first and followed by the payload of all messages, it will lead to an optimization where we have to "partially" decompress the batch (i.e. the prefix).

Ideas for message format v.3 wiki: ideas for kafka message format v.3

Motivation

Logs in kafka consists of batches, and one batch consists of many messages. So if the size of each message header can be reduced, it'll improve the network traffic and storage size (and money, of course) a lot, even it's a small improvement. Some companies now handle trillions of messages per day using Kafka, so, if we can reduce just 1 byte per message, the save is really considerable.

Currently, The message format in V2 is like this:

```
length: varint
attributes: int8
    bit 0~7: unused // <-- unused attributes field
timestampDelta: varlong
offsetDelta: varint
keyLength: varint
key: byte[]
valueLen: varint
value: byte[]
Headers => [Header]
```

We can focus on the `attributes` field, it is introduced since message format v1 and till now, it is still unused.

So, I'm proposing we can add a flag in batch attribute to indicate if the messages inside the batch have attributes field or not.

Public Interfaces

In the record batch header, We'll add a new attribute bit to indicate if the messages in the batch contain attributes or not. If it's 0, it means the messages have attribute field (like what we have now). If it's set to 1, it means no attribute field. Also, the magic value will bump to 3.

```
baseOffset: int64
batchLength: int32
partitionLeaderEpoch: int32
magic: int8 (current magic value is 2) // <-- will bump to 3
crc: int32
attributes: int16
    bit 0~2:
        0: no compression
        1: gzip
        2: snappy
        3: lz4
        4: zstd
    bit 3: timestampType
    bit 4: isTransactional (0 means not transactional)
    bit 5: isControlBatch (0 means not a control batch)
    bit 6: hasDeleteHorizonMs (0 means baseTimestamp is not set as the delete horizon for compaction)
    // new added attribute below
    bit 7: ignoreMessageAttributes (0 means not to ignore)
    bit 8~15: unused

producerId: int64
producerEpoch: int16
baseSequence: int32
lastOffsetDelta: int32 // <-- will change to varint
baseTimestamp: int64 // <-- will change to varlong
maxTimestamp: int64 // <-- will change semantic to maxTimestampDelta and change type to varint records: [Record]
```

Furthermore, the record batch header can also be smaller. I'd also like to improve them by:

1. **baseTimestamp**: change type from int64 to varlong. With varlong, our current timestamp in ms needs only 6 bytes. I've tested, it still needs only 6 bytes after 10 years.

2. **maxTimestamp**: change the semantic to **maxTimestampDelta**, and change type from int64 to varint. In most case, the timestamp for each record inside the batch should be very close. So, changing to varint will save space. The maxTimestamp can be calculated via [baseTimestamp + maxTimestampDelta]

3. **lastOffsetDelta**: change the type from int32 to varint. Same as above, In most case, the offset delta should be small. So, changing to varint will save space.

4. move above 3 fields to the trail of the batch headers.

Note:

The biggest value of varint 2 bytes is 16383

The biggest value of varint 3 bytes is 2097151

With the above 3 record batch fields changes, in a normal batch, with close timestamp between each record and offsets, the save can be:

MaxTimestamp from int64 (8 bytes) to 2 ~ 3 bytes. Suppose the max timestamp delta has a long 16 seconds, which only need 2 bytes for varint. The **offset delta** save from int32 (4 bytes) to 1~2 bytes (

In all, we can save around:

[8 - 6 (baseTimestamp)] + [8 - 2 (max timestamp delta)] + [4 - 2 (offset delta)] = 10 Bytes for each batch.

18% space save for batch overhead, compared with original batch overhead 53 bytes.

For the each message header space save, using the example in KIP-98:

For example, assuming a fixed message size of 1K with 100 byte keys and reasonably close timestamps, the overhead increases by only 7 bytes for each additional batched message (2 bytes for the message size, 1 byte for attributes, 2 bytes for timestamp delta, 1 byte for offset delta, and 1 byte for key size)

14% space save for each message overhead, compared with originally 7 bytes, now only 6 bytes needed.

Proposed Changes

When writing the batch using the new version of message format, we'll default set ignoreMessageAttributes field to 1 in record batch header, and create records without attribute field.

When reading batches, we'll first check the ignoreMessageAttributes field before reading each records, and read messages accordingly.

When old consumer reading the new formatted message, we'll do a downconvert to the older message format like what we did for old consumer accepting message format v1 now.

Compatibility, Deprecation, and Migration Plan

Totally backward compatible. Even if one day we want to use message attribute field, we can always update the ignoreMessageAttributes to add attributes.

Test Plan

Unit test + Integration test

Rejected Alternatives

Keep using the existing message format. But that would keep wasting 1 byte each message to store zero content.