KIP-31 - Move to relative offsets in compressed message sets

- Status
- Motivation
- Public Interfaces
- Proposed Changes
    - Wire protocol change
        - Change the usage of offset field
        - Add a message.format.version configuration to the broker
- Compatibility, Deprecation, and Migration Plan
- Rejected Alternatives

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *KAFKA-2511*

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

Motivation

Today the broker needs to decompress compressed messages, assign offsets to each message and recompress the messages again. This causes additional CPU cost. This KIP is trying to avoid server side recompression.

This KIP is a distilled/improved version of an earlier discussion that we started.

Public Interfaces

We propose the following change to the message format

```
MessageAndOffset => Offset MessageSize Message
  Offset => int64 <------ CHANGE: Base offset for wrapper message of compressed message, relative offset for
inner compressed message.
  MessageSize => int32

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

Proposed Changes

Wire protocol change

Change the usage of offset field

- When the producer compresses a 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
    1. Decompress the message to verify the CRC and relative offset.
       NOTE: If the relative offset is not contiguous (e.g., if this is a mirrored compacted topic), the broker will reassign the relative offsets. There are two ways to handle this - (i) reject the ProducerRequest or (ii) just assign the relative offsets. We chose to reassign offsets rather than reject the request because there is a useful use case where mirror maker can do a direct copy from source cluster to

destination cluster without even decompressing the message. In this case, the compressed message can have noncontinuous relative offsets (for compacted topics).
    2. Set outer message's base offset. The outer message's base offset will be the offset of the last inner message. (Since the broker only needs to update the message-set header, there is no need to re-compress message sets.)
- When the log cleaner compacts log segments, if multiple message sets are compacted into one message set, the broker needs to update the inner message's relative offset values. (This will leave "holes" inside the new wrapped message).
- When the consumer receives a message, it converts the relative offset back to actual offset.

Add a message.format.version configuration to the broker

- The message.format.version controls the message format written to disk. The value equals to the magic byte of the message format. Introducing this configuration is to avoid doing version up/down conversion for the majority of users.
- If a consumer supports message.format.version, the broker will just use zero-copy transfer to send back the FetchResponse.
- If a consumer does not support message.format.version, the broker will have to do down conversion and send FetchResponse without using zero-copy.

Compatibility, Deprecation, and Migration Plan

NOTE: This part is drafted based on the assumption that KIP-31 and KIP-32 will be implemented in one patch.

The proposed protocol is not backward compatible. The migration plan are as below:

Phase 1 (MessageAndOffset V0 on disk):

1. Set message.format.version=0 on brokers. (Broker will write MessageAndOffset V0 to disk)
2. Create internal ApiVersion 0.9.0-1** which uses ProducerRequest V2 and FetchRequest V2.
3. Configure the broker to use ApiVersion 0.9.0 (ProduceRequest V1 and FetchRequest V1).
4. Do a rolling upgrade of the brokers to let the broker pick up the new code supporting ApiVersion 0.9.0-1.
5. Bump up ApiVersion of broker to 0.9.0-1. to let the broker use FetchRequest V2 for replication.
6. Upgraded brokers support both ProducerRequest V2 and FetchRequest V2 which uses magic byte 1 for MessageAndOffset.
    a. When broker sees a producer request V1 (MessageAndOffset = V0), it will decompress the message, assign offsets using absolute offsets and re-compress the message.
    b. When broker sees a producer request V2 (MessageAndOffset = V1), it will decompress the message, assign offsets using absolute offsets and do re-compression. i.e. down-convert the message format to MessageAndOffset V0.(This is no worse than what the brokers are currently doing.)
    c. When broker sees a fetch request V1 (Supporting MessageAndOffset = V0), because the data format on disk is MessageAndOffset V0, it will use the zero-copy transfer to reply with fetch response V1 with MessageAndOffset V0.
    d. When broker sees a fetch request V2 (Supporting MessageAndOffset = V0, V1), because the data format on disk is MessageAndOffset V0, it will use zero-copy transfer to reply with fetch response V2 with MessageAndOffset V0.
7. Upgrade consumer to send FetchRequest V2.
8. Upgrade producer to send ProducerRequest V2.

Phase 2 (MessageAndOffset V1 on disk):

1. After most of the consumers are upgraded, Bump up message.format.version=1 and rolling bounce the brokers.
2. Upgraded brokers do the followings:
    a. When broker sees a producer request V1 (MessageAndOffset = V0), it will decompress the message, assign offsets using relative offsets and re-compress the message. i.e. up-convert the message format to MessageAndOffset V1.
    b. When broker sees a producer request V2 (MessageAndOffset = V1), it will decompress the message, assign offsets using relative offsets and NOT do re-compression.
    c. When broker sees a fetch request V1 (Supporting MessageAndOffset = V0), because the data format on disk is MessageAndOffset V1, it will NOT use the zero-copy transfer. Instead the broker will read the message from disk, down-convert them to V0 and reply using fetch response V1 with MessageAndOffset V0.
    d. When broker sees a fetch request V2 (Supporting MessageAndOffset = V0, V1), because the data format on disk is MessageAndOffset V1, it will use zero-copy transfer to reply with fetch response V2 with MessageAndOffset V1.

For producer, there will be no impact.

In phase 1, there will be no impact for consumers.

In phase 2, there will be some performance penalty for consumers that only supports MessageAndOffset V0, because there is no zero-copy transfer.

At the beginning of phase 2, there will be some time the log segment contains both MessageAndOffset V0 and V1. The broker will always do down conversion for FetchRequest V1 and zero-copy transfer for FetchRequest V2.

** We introduce internal ApiVersion here to help the user who are running on trunk to upgrade in the future. Otherwise the interim ApiVersion between two official releases will require users to downgrade ApiVersion then upgrade.

To canary a broker

After phase 1, it is possible for user to canary a broker in phase 2 and roll back if something goes wrong. The procedure is:

1. Set message.format.version=1 on one of the brokers (broker B).
2. Broker B will start to act like what described in phase 2.
    a. It will sends FetchRequest V2 to other brokers for replication.
    b. It will only see ProduceRequest/FetchRequest V1 from other brokers and clietns.
3. If something goes wrong, we can do the following to rollback:

a. shutdown broker B
b. nuke the data of the topics it was serving as leader before shutdown
c. set message.format.version=0
d. restart the broker to let the broker replicate from leaders. At this point the data on disk will be in MessageAndOffset V0.

In step 2, it is recommended to put only small amount of leaders on the broker, because at that point the broker needs to do down conversion for all the fetch requests.

Rejected Alternatives

None