KIP-126 - Allow KafkaProducer to split and resend oversized batches.

Status

Current state: *Accepted*

Discussion thread: *here*

JIRA: *KAFKA-3995*

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

Motivation

Kafka has a strict message size limit on the messages. This size limit is applied to the compressed messages as well.

Currently KafkaProducer uses an estimation to do guess the compressed message size from the uncompressed message size. The estimation is based on a weighted average in a sliding window on the compression ratio of the most recent batches for each compression type. The formula is the following:

Assume `COMPRESSION_RATIO_N` stands for the compression ratio of the `N`th batch. The estimated compression ratio for the `(N+1)`th batch is:

```
(COMPRESSION_RATIO_N * DAMPING_FACTOR^(N - 1) * (1 - DAMPING_FACTOR)) + INITIAL_COMPRESSION_RATIO * DAMPING_FACTOR^N
```

When the `(N+1)`th batch is generated, this estimated compression ratio will be used (multiplied by a factor of 1.05 for contingency) to estimate the compressed size from the uncompressed size. When the estimated compressed size reaches the batch.size configuration, the batch will be closed and sent to the brokers.

The problem of the current behavior is that this estimation could be off and cause RecordTooLargeException.

For example, if the batch size is set to 1MB and the max message size is 1MB. Initially a the producer is sending messages (each message is 1MB) to topic_1 whose data can be compressed to 1/10 of the original size. After a while the estimated compression ratio in the compressor will be trained to 1/10 and the producer would put 10 messages into one batch. Now the producer starts to send messages (each message is also 1MB) to topic_2 whose message can only be compress to 1/5 of the original size. The producer would still use 1/10 as the estimated compression ratio and put 10 messages into a batch. That batch would be 2 MB after compression which exceeds the maximum message size. In this case the user do not have many options other than resend everything or close the producer if they care about ordering and message loss.

This is especially an issue for services like MirrorMaker whose producer is shared by many different topics.

This KIP proposes to solve this issue by doing the followings:

1. Change the way to estimate the compression ratio
2. Split the oversized batch and resend the split batches.

Public Interfaces

This KIP introduces the following new metric `batch-split-rate` to the producer. The metric records rate of the batch split occurrence.

Although there is no other public API change, due to the behavior change, users may want to reconsider batch size setting to improve performance.

Proposed Changes

Decompress the batch which encounters a RecordTooLargeException, split it into two and send it again

There are a few things to be think about for this approach:

1. More overhead introduced on the producer side. The producer have to decompress the batch, regroup the messages and resend them. If the producer keeps the original uncompressed message to avoid potential decompression, it will have huge memory overhead.

2. The split batches is not guaranteed to be smaller than the max size limit. Potentially there will multiple retries until the messages get through, or fail.
3. In the scenario such as mirror maker, due to different compression ratio in different topics, some of the topics may have very different compression ratio from the average compression ratio, this will potentially introduce many split and resend, which introduces a lot of overhead.

To address the above caveats, we propose to also change the way to estimate the compression ratio:

1. Estimate the compression ratio for each topic independently.
2. Given that `COMPRESSION_RATIO = COMPRESSED_SIZE/UNCOMPRESSED_SIZE`, Change the compression ratio estimation from weighted average on a sliding window to the following:
    a. Initially set `ESTIMATED_RATIO = 1.0`
    b. If `OBSERVED_RATIO < ESTIMATED_RATIO`, decrease the `ESTIMATED_RATIO` by `COMPRESSION_RATIO_IMPROVING_STEP (0.005)`
    c. If `OBSERVED_RATIO > ESTIMATED_RATIO`, increase the `ESTIMATED_RATIO` by `COMPRESSION_RATIO_DETERIORATE_STEP (0.05)`
    d. When estimating the batch size, the producer will use (`ESTIMATED_RATIO` * `UNCOMPRESSED_SIZE` * 1.05) where 1.05 is the `COMPRESSION_RATIO_ESTIMATION_FACTOR` for contingency.
    e. If batch split occurred, reset the `ESTIMATED_RATIO` to 1.0

Based on the test in this patch, the chance of splitting a batch is much less than 10% even when the compression ratio of the messages in the same topic are highly different.

NOTE: The `COMPRESSION_RATIO_IMPROVING_STEP, COMPRESSION_RATIO_DETERIORATE_STEP` and `COMPRESSION_RATIO_ESTIMATION_FACTOR` are sort of magic number but chosen based on the following considerations:

1. The time for the estimated compression ratio to approach the observed compression ratio.
    - The larger the `COMPRESSION_RATIO_IMPROVING_STEP` is, the faster the estimated compression ratio will approach the observed compression ratio range.
2. How stable the estimated compression ratio is:
    - The larger the `COMPRESSION_RATIO_DETERIORATE_STEP` and `COMPRESSION_RATIO_IMPROVING_STEP` are, the more churn will there be.
3. Reduce the likelihood of batch split.
    - The larger the `COMPRESSION_RATIO_DETERIORATE_STEP:COMPRESSION_RATIO_IMPROVING_STEP` is, the less batches would be split.
    - The smaller `COMPRESSION_RATIO_IMPROVING_STEP` is, the less batches would be split.
    - The larger the `COMPRESSION_RATIO_ESTIMATION_FACTOR` is, the less likely a batch will be split.
4. Efficiency
    - The larger the `COMPRESSION_RATIO_ESTIMATION_FACTOR` is, the less efficient the compression would be.

The effect of those three values are difficult to quantify on per message basis, and would vary according to the traffic pattern. However, here are how they are chosen:

1. Choosing `COMPRESSION_RATIO_DETERIORATE_STEP` to be 0.005 means that it will take about 20 batches to decrease compression ratio by 0.1. So in a normal case, after a batch is split, it takes quite a few batches to reach another good estimated compression ratio which may cause batch split again (assuming there is no batch in the middle to deteriorate the compression ratio estimation).
2. Use 10:1 as `COMPRESSION_RATIO_DETERIORATE_STEP:COMPRESSION_RATIO_IMPROVING_STEP.` As an intuitive approximation, statistically speaking this allows the number of high compression ratio message and low compression ratio message to be less than 10:1 without a split. For example, when there are 9 batches improving the compression ratio, the 10th batch may deteriorate the compression ratio.
3. For compression efficiency, 1.05 is chosen as the `COMPRESSION_RATIO_ESTIMATION_FACTOR`. This value is what we are using today and it gives reasonable contingency for batch size estimation.

Compatibility, Deprecation, and Migration Plan

The KIP is backwards compatible.

Rejected Alternatives

Batching the messages based on uncompressed bytes

Introduce a new configuration **enable.compression.ratio.estimation** to allow the users to opt out the compression ratio estimation, but use the uncompressed size for batching directly.

The downside of this approach are:

1. it adds a new configuration to the producer which exposes some nuances.
2. for highly compressible messages, users may still need to guess the compression ratio to ensure the compressed batch is not too small.

Splitting Batches based on configured batch.size

The concern for that is that batch size is not only associated with the max message size, but also related to the memory consumption. For example, if a producer is sending messages to 1000 partitions (Mirror Maker usually has a much larger number) and max message size is 1 MB, setting the batch size to max message size means it will take 1 GB memory to hold one batch for each partition. This would actually result in unwanted small batches because the batches need to be sent out prematurely to release memory for the new batch creation.

So in practice, we usually set the batch size to a number that it has good batching but do not require too much memory to be allocated to a producer, say 500 KB. In this case, we don't want to unnecessarily split the batch even if it is bigger than the configured batch size because likely it will still be less than the max message size.
It is true that if the batch size is set to be larger than max message size, the performance will drop significantly and users may not see any exception. But in practice this would be more of a misconfiguration. The ideal solution to this would be letting the producer get the max message size configuration from the broker and split the batches based on that before sending it over the wire. This will also avoid the misconfiguration. For now since we have the metrics for user to detect frequent batch split, it is a good intermediate stage before we have the ideal solution.