KIP-782: Expandable batch size in producer

Status

Current state: *"Voting"*

Discussion thread: *here*

JIRA: *here*

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

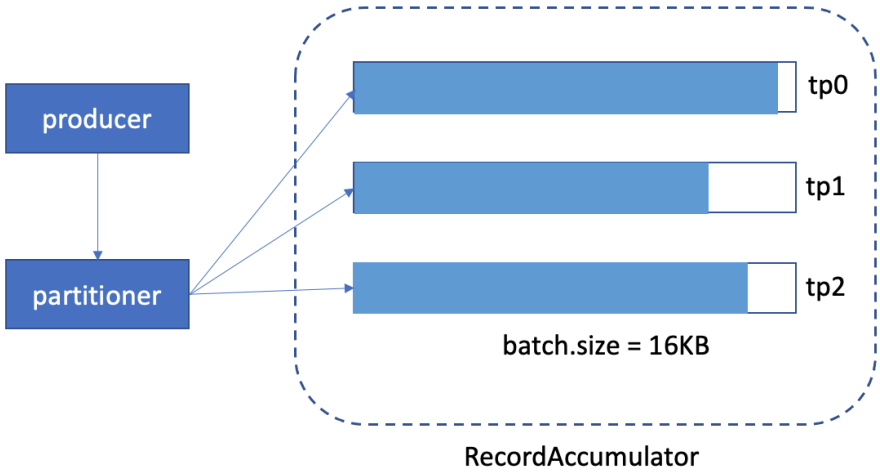
Motivation

Kafka provides high throughput in each component. In the producer, there are 2 key configs to support the high throughput:

1. **batch.size**: it specifies a maximum batch size in bytes per partition (default 16384)
2. **linger.ms**: it specifies a maximum duration to fill the batch in milliseconds (default 0 or no delay)

Records are sent until either of above 2 thresholds are reached.

The high level producer component diagram is like this:



However, when we set the batch size, we'll run into a dilemma:

1. either we have higher throughput + more memory waste,
2. or we have slower throughput + less memory waste.

Why is that?

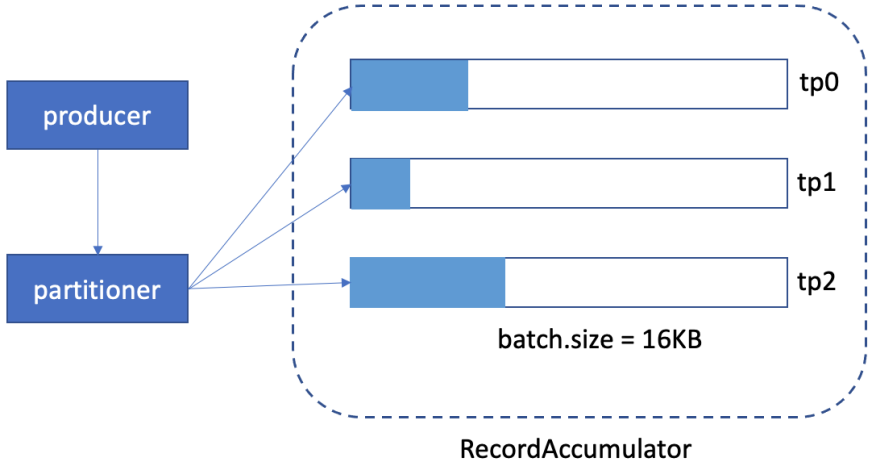
We can check the description for "batch.size" in documentation here. Here's the last paragraph I extracted from the document:

A small batch size will make batching less common and may reduce throughput (a batch size of zero will disable batching entirely). A very large batch size may use memory a bit more wastefully as we will always allocate a buffer of the specified batch size in anticipation of additional records.

That explains why we have the dilemma when setting this config.

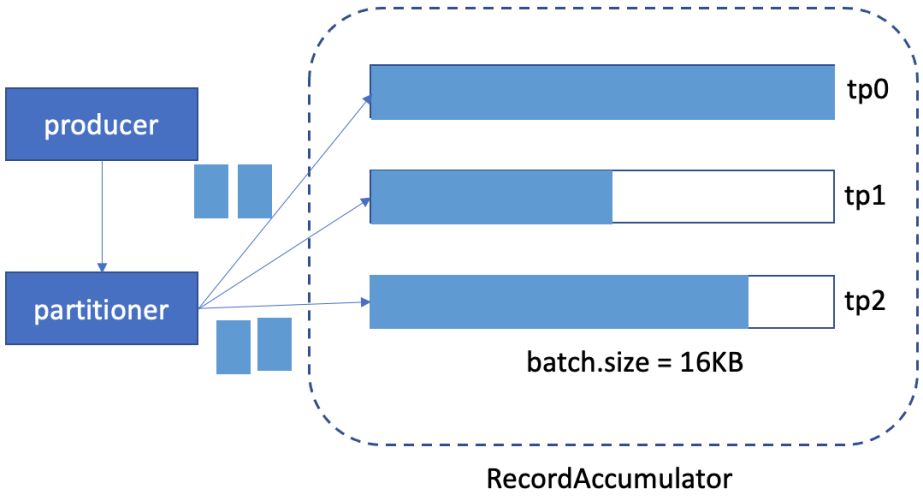
Use the above example, we set the "batch.size" as default 16KB.

1. off-peak time, the graph might look like this(when linger.ms expired):



We can see, the memory usage is very low.

2. peak time, the graph might look like this:



We can see, the tp0 batch is full, and need to create new batch for it and send out this batch soon.

In the peak-time case, when we notice this situation, we might want to increase the batch size for it, let's say, increase from 16KB to 20KB. But when we increase the batch.size, we know all batches will be allocate with 20KB from now on, even it's off-peak time (check case 1). That's wasteful.

Furthermore, currently batch.size is used in two conditions:

1. When we append records to a batch in the accumulator, we create a new batch if the current batch would exceed the batch.size.
2. When we drain the batch from the accumulator, a batch becomes 'ready' when it reaches batch.size.

The second condition is good with the current batch size, because if linger.ms is greater than 0, the send can be triggered by accomplishing the batching goal.

The first condition, though, leads to creating many batches if the network latency or production rate (or both) is high, and with 5 in-flight and 16KB batches we can only have 80KB of data in-flight per partition. Which means that with 50ms latency, we can only push ~1.6MB/sec per partition (this goes down if we consider higher latencies, e.g. with 100ms we can only push ~0.8MB/sec).

In this KIP, I'm going to introduce a dynamic expandable buffer size for producer, with a larger allowable batch size (i.e. "batch.max.size").

Goal:

1. **higher throughput**: with the "batch.max.size" introduced, the batch can allow ("batch.size" < records < "batch.max.size") to be sent within one batch. So, even a sudden high producer rate, the throughput can still be good.
2. **better memory usage**: with "batch.initial.size" introduced, the initial memory allocation will be small. So when setting the "batch.size", you don't have to consider the memory waste issue anymore.

Public Interfaces

2 Producer config will be introduced:

1. **batch.initial.size**: It specifies the initial batch size when new batch created, it should be <= "batch.size" (default is 4096(4KB))
2. **batch.max.size**: It specifies the maximum batch size in bytes per partition, it should be <= "max.request.size" and >= "batch.size". ("batch.max.size" default to 262144(256KB))

To have a better memory usage, the relation of the configurations **should** be:

1. "batch.size" = "batch.initial.size" * n (n means the times we expansion)
2. "batch.max.size" = "batch.initial.size" * m (m means the times we expansion)

ex:

1. "batch.size" = 16KB, "batch.initial.size" = 4KB

=> 16KB = 2KB * 4

2. "batch.size" = 16KB, "batch.initial.size" = 4KB, "batch.max.size" = 256KB

=> 256KB = 4KB * 64

Proposed Changes

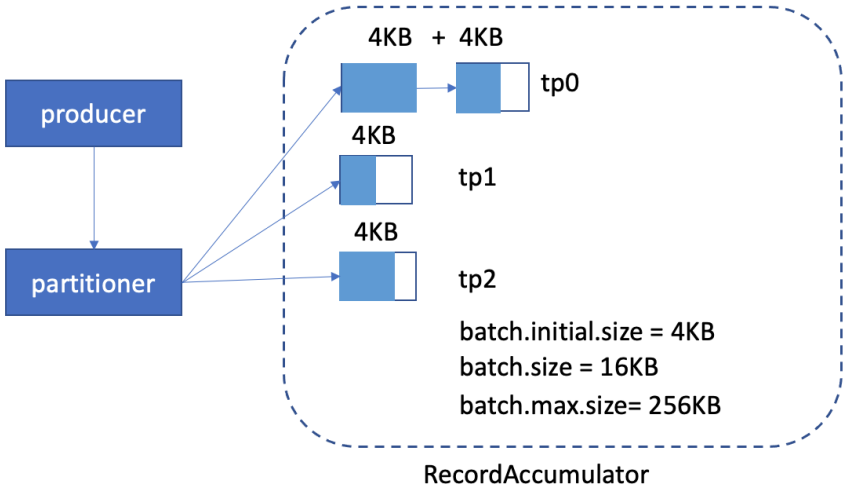
We'll allocate the "batch.initial.size" (default 4KB) memory when new records send to an empty partition buffer. While we accumulated more records in the partitions to reach the "batch.initial.size" (4KB), we'll do buffer expansion to allocate another "batch.initial.size" of buffer and list with the previous buffer. And then, keeps accumulating the records.

As current producer, when the batch reaching the "batch.size", it means the buffer is "ready" to be be sent. But now, before the sender thread is ready to send the batch, if there are more data coming, we can still accumulate it into the same buffer, until it reached the "batch.max.size" or sender thread send this batch out. After it reached the "batch.max.size", we'll create another batch for it. Compared with current producer, when reaching the "batch.size", we'll mark it as "ready" to be sent, and create a new batch for the upcoming records.

Internally, we have a BufferPool#free to keep the unused allocated buffers. When allocating a buffer, we'll first check if there are available buffer there before allocating a new one. With the expandable batch introduced, there are chances that sudden high producer rate, there are a lot of buffers being allocated, and then send back to BufferPool unused. Some of the buffers in BufferPool might not be used for a long time. In this KIP, when sending the allocated batches back to BufferPool#free after batch sent, we'll only keep maximum "batch.size" into pool, and mark the rest of memory as free to use. The reason we keep maximum "batch.size" back to pool is because the semantic of "batch.size" is the batch full limit. In most cases, the batch.size should be able to contain the records to be sent within linger.ms time.

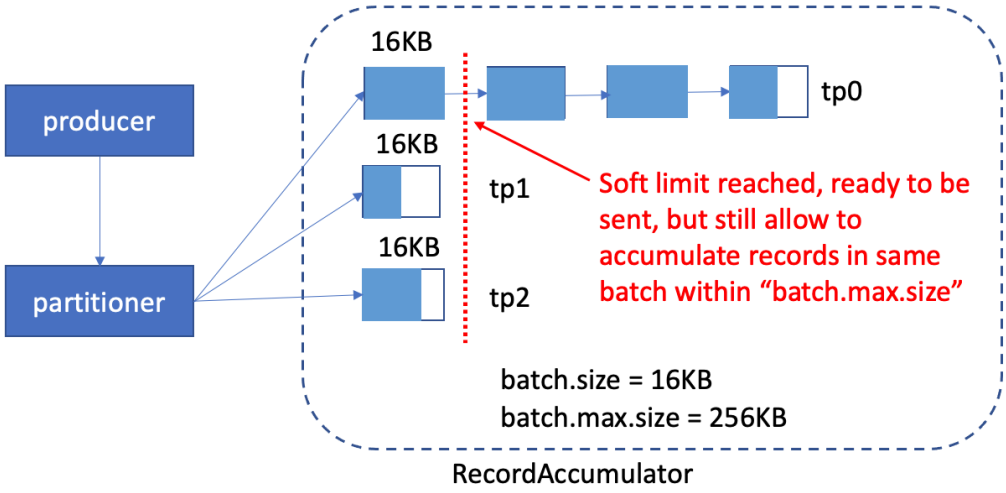
So, let's see the 2 cases above

1. off-peak time



We can see now, the memory usage is still high, because we allocate batch.initial.size(4KB) each chunk.

2. peak-time



With the batch.max.size config introduced, we can have a larger amount of batch before the sender thread is ready to send the batch, to achieve high throughput.

Compatibility, Deprecation, and Migration Plan

2 new config for batch size introduced in this KIP. There will be always backward compatible. So no migration plan is needed.

Rejected Alternatives

- Do we really need batch.initial.size? It's not clear that having this extra setting adds a lot of value.

--> I made the default value to 4KB now, which means it'll make the producer's memory usage better after upgrading to the new release.