

# Producer Performance Tuning For Apache Kafka

Jiangjie (Becket) Qin @ LinkedIn

Streams Meetup @ LinkedIn  
June 15, 2015

# What is Apache Kafka?

“ Kafka is a **high throughput low latency** ..... ”

Performance tuning is

- still very important!
- a case by case process based on
  - Different data pattern
  - Performance objectives

# Agenda

- The goal of producer performance tuning
- Understand Kafka Producer
- Producer performance tuning
  - ProducerPerformance tool
  - Quantitative analysis using producer metrics
  - Play with a toy example
- Some real world examples
  - Latency when acks=-1
  - Produce when RTT is long
- Q & A

# Agenda

- **The goal of producer performance tuning**
- Understand Kafka Producer
- Producer performance tuning
  - ProducerPerformance tool
  - Quantitative analysis using producer metrics
  - Play with a toy example
- Some real world examples
  - Latency when acks=-1
  - Produce when RTT is long
- Q & A

# The goal of producer performance tuning

With a **given dataset** to send:

- Achieve the **throughput** and **latency** goals with guarantees of
  - Durability
  - Ordering
- Focus on the average producer performance
  - 99 percentile performance sometimes cannot be “tuned”
  - Tuning average performance also helps 99 percentile numbers
- Today’s talk
  - Fixed data pattern (1000 bytes of integers ranging between 0 - 50000)

# Agenda

- The goal of producer performance tuning
- **Understand the Kafka Producer**
- Producer performance tuning
  - ProducerPerformance tool
  - Quantitative analysis using producer metrics
  - Play with a toy example
- Some real world examples
  - Latency when acks=-1
  - Produce when RTT is long
- Q & A

# Understand the Kafka producer

- **`org.apache.kafka.producer.KafkaProducer`**
  - If you are still using OLD producer, please upgrade!
- Benchmarks in this talk use Kafka 0.10.0
  - No broker side recompression ([KIP-31](#))
  - 8 bytes overhead per message due to the introduction of timestamp ([KIP-32](#))

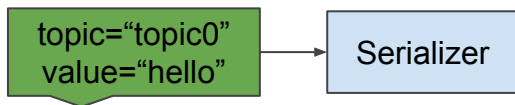
# Critical Configurations

- batch.size
- linger.ms
- compression.type
- max.in.flight.requests.per.connection (*affects ordering*)
- acks (*affects durability*)



# Understand the Kafka producer

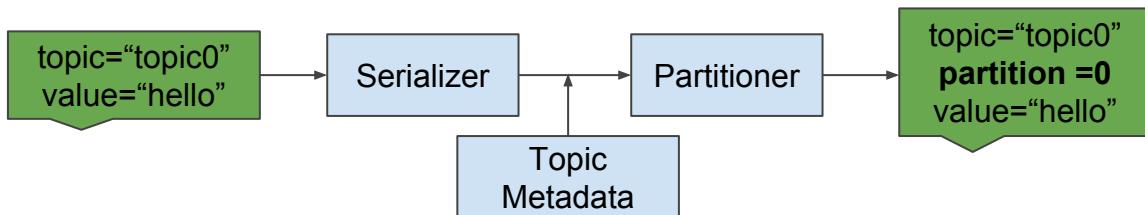
**User:** `producer.send(new ProducerRecord("topic0", "hello"), callback);`



- Serialization

# Understand the Kafka producer

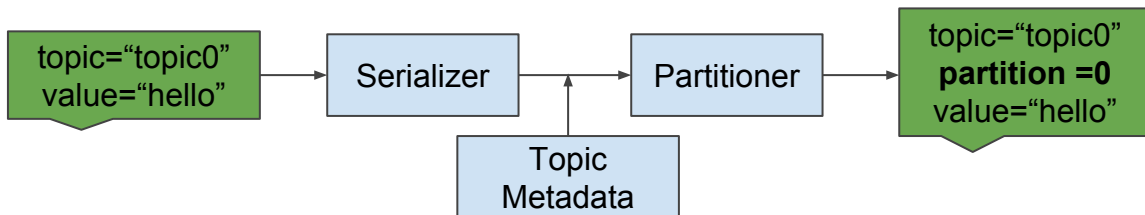
**User:** `producer.send(new ProducerRecord("topic0", "hello"), callback);`



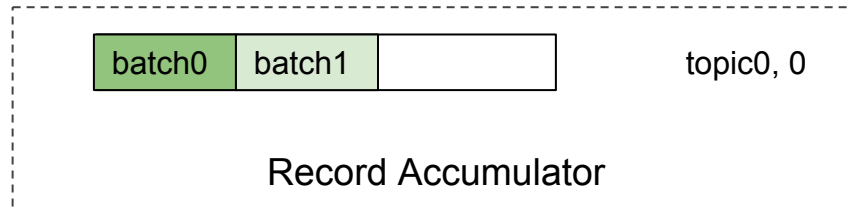
- Serialization
- Partitioning

# Understand the Kafka producer

**User:** `producer.send(new ProducerRecord("topic0", "hello"), callback);`

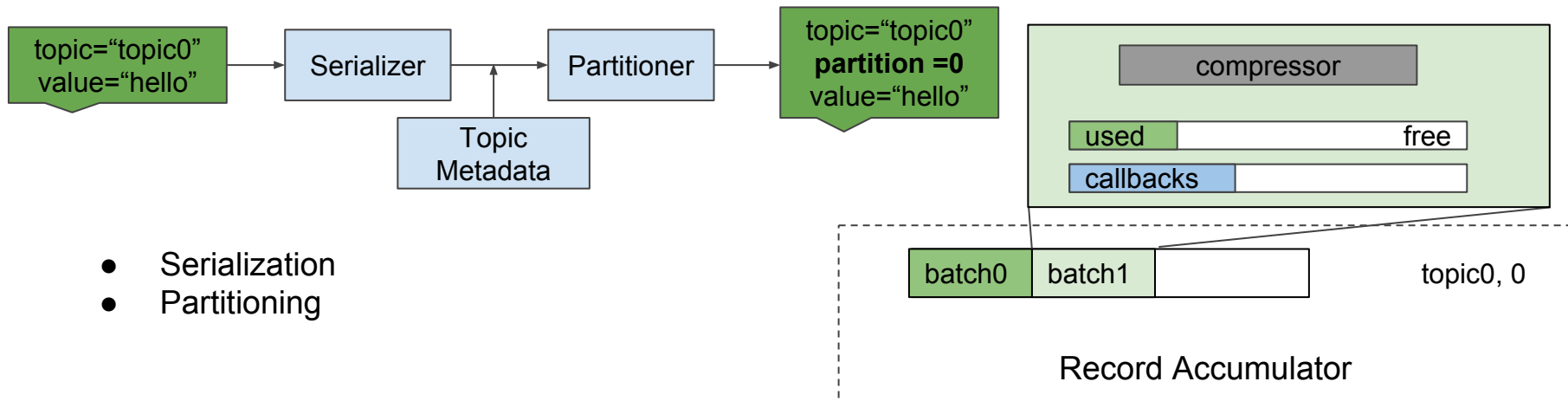


- Serialization
- Partitioning



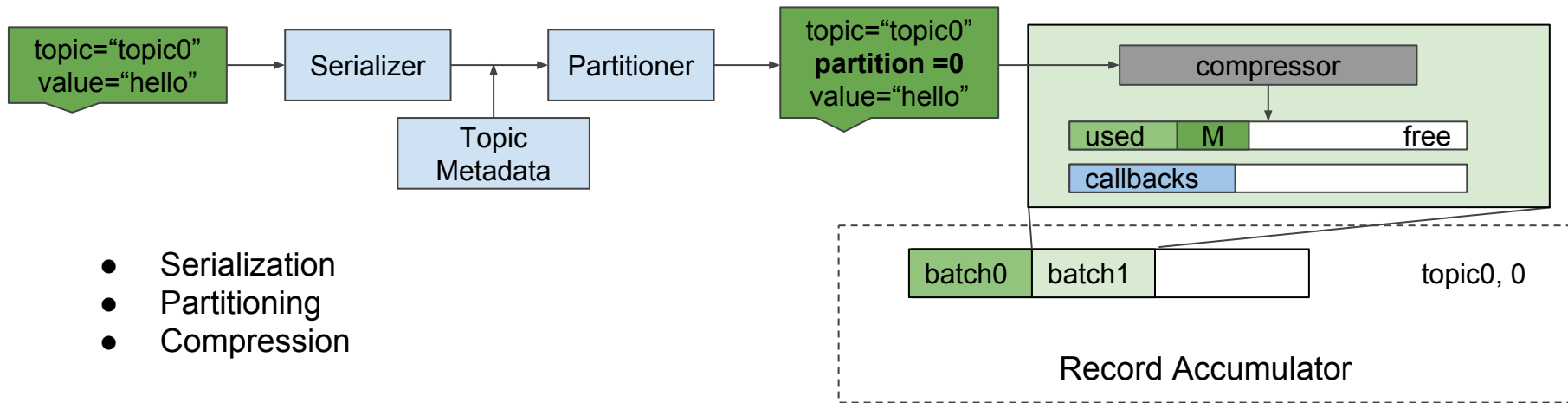
# Understand the Kafka producer

**User:** `producer.send(new ProducerRecord("topic0", "hello"), callback);`



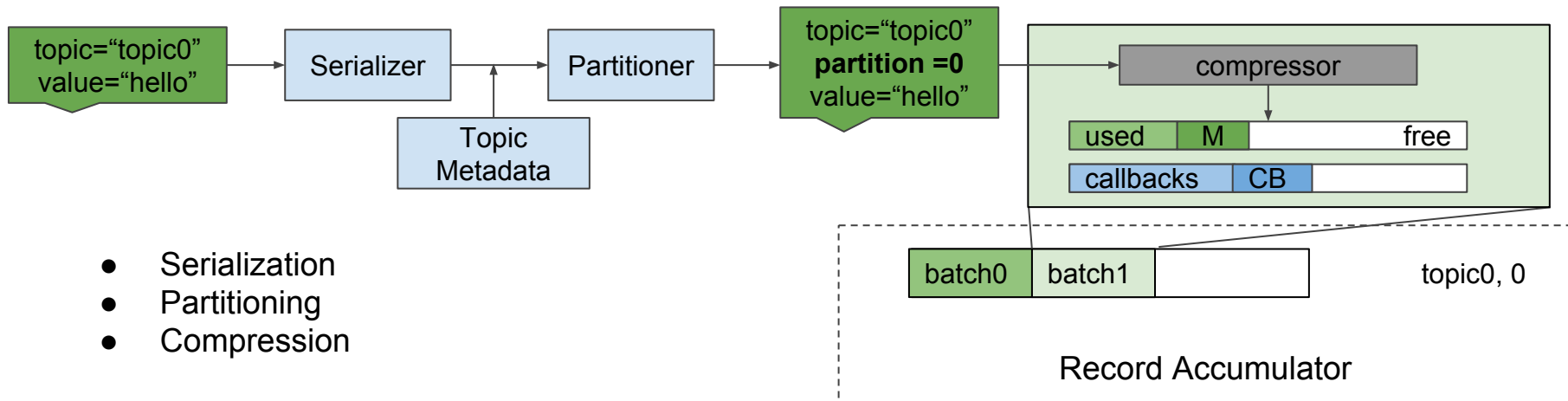
# Understand the Kafka producer

**User:** `producer.send(new ProducerRecord("topic0", "hello"), callback);`



# Understand the Kafka producer

**User:** `producer.send(new ProducerRecord("topic0", "hello"), callback);`



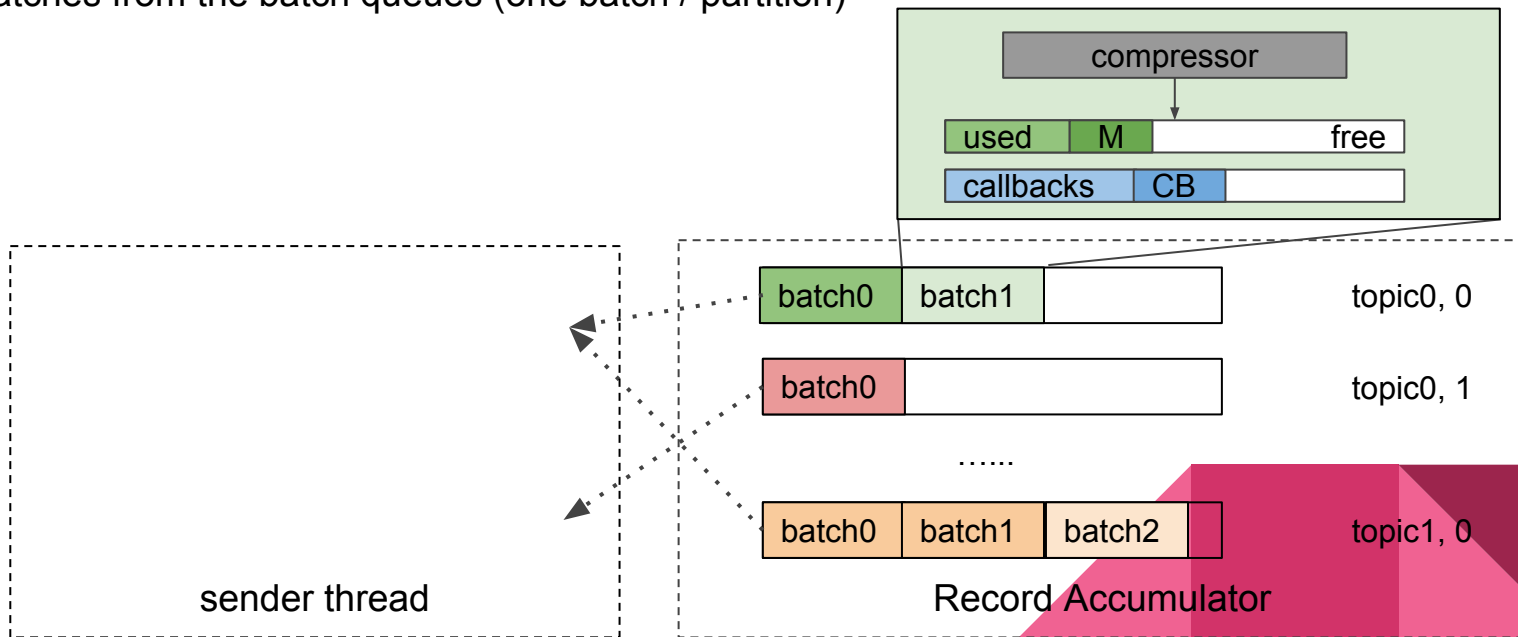
- Serialization
- Partitioning
- Compression

Tasks done by the user threads.

# Understand the Kafka producer

## Sender :

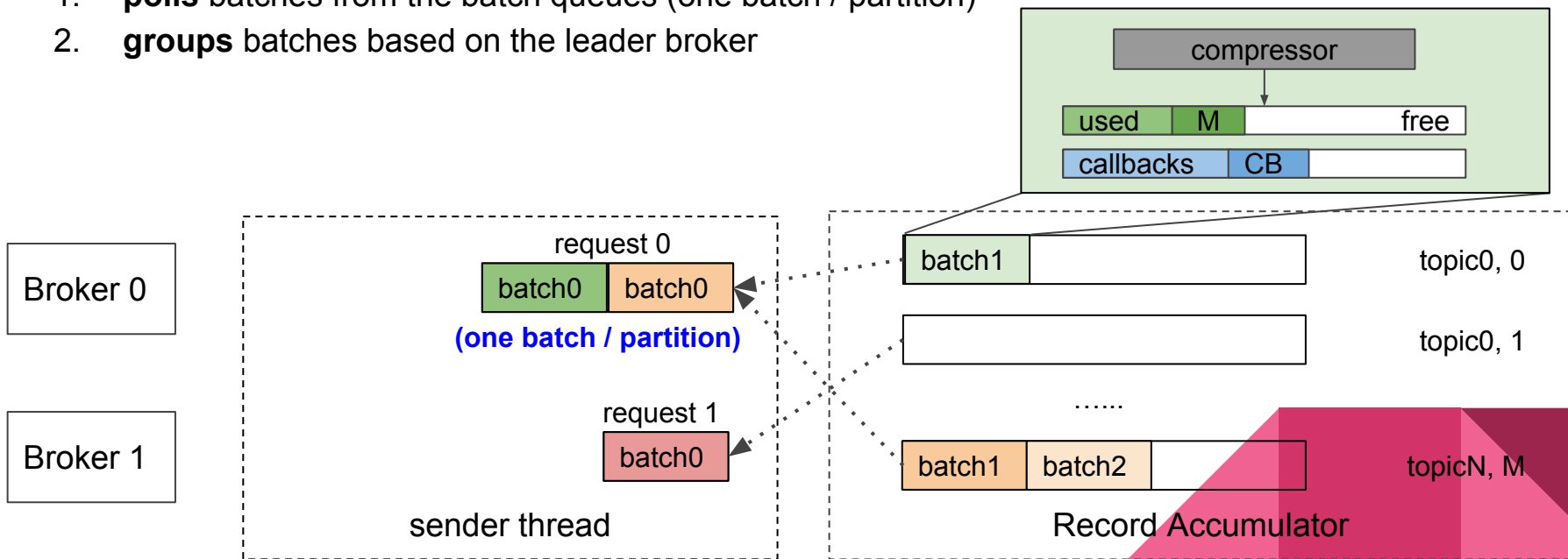
1. **polls** batches from the batch queues (one batch / partition)



# Understand the Kafka producer

## Sender :

1. **polls** batches from the batch queues (one batch / partition)
2. **groups** batches based on the leader broker

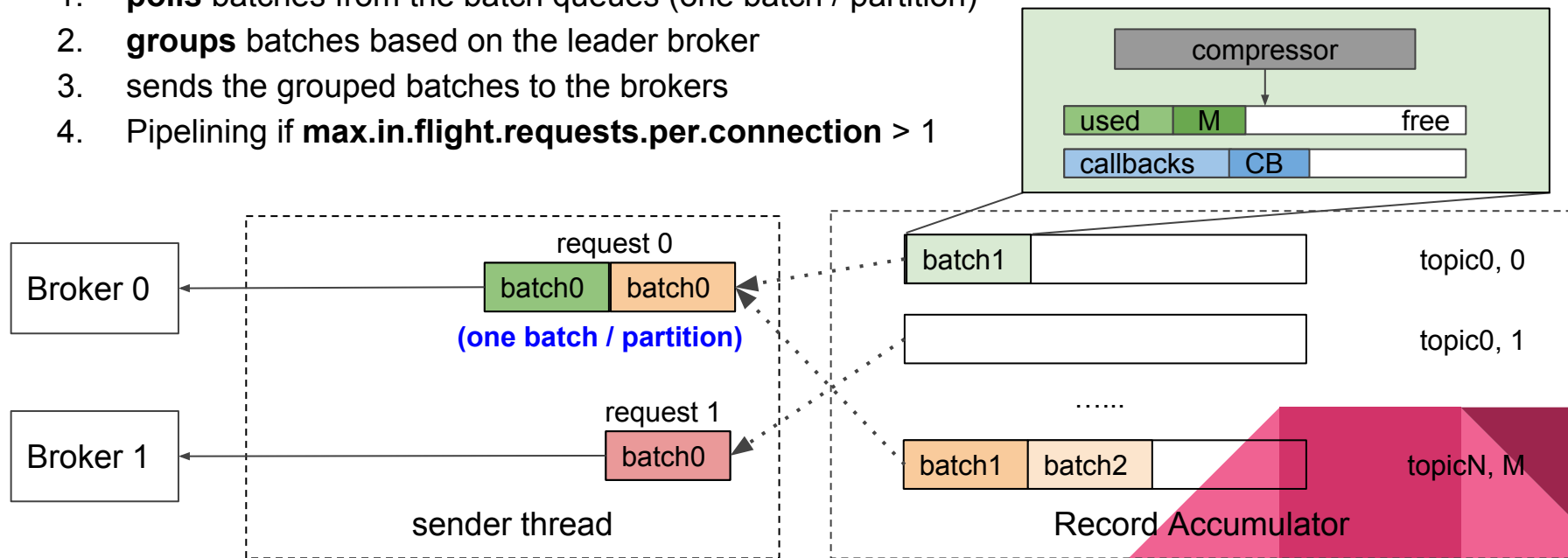




# Understand the Kafka producer

## Sender :

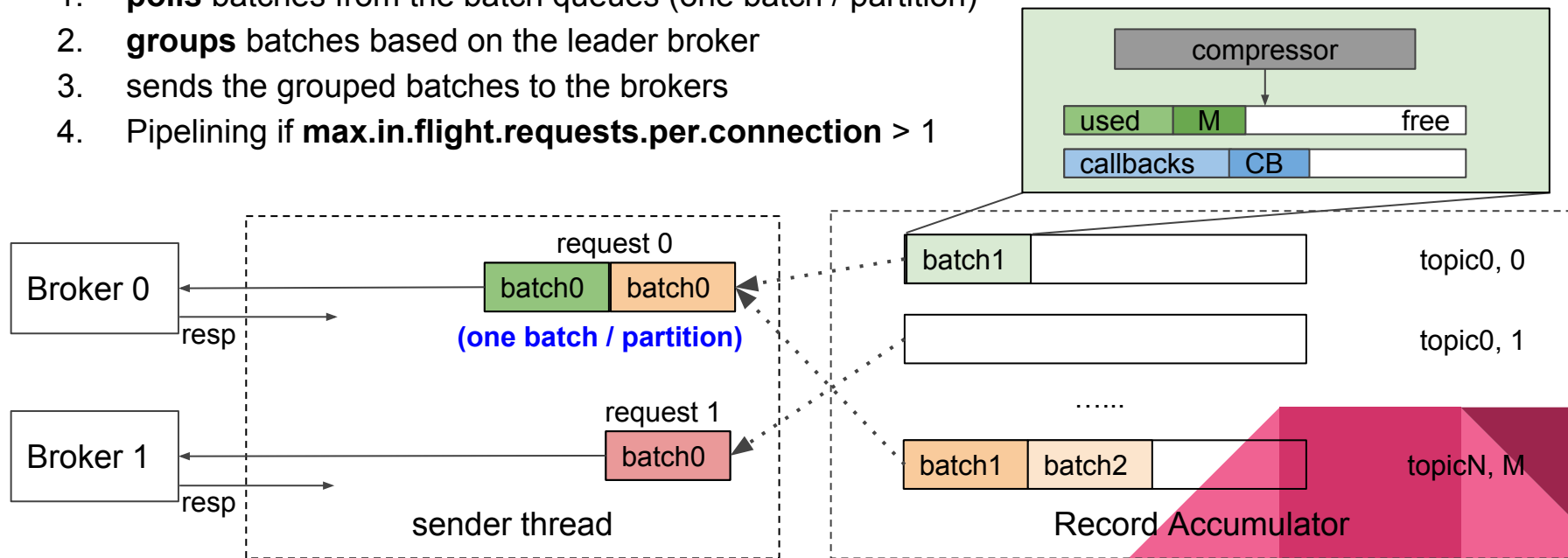
1. **polls** batches from the batch queues (one batch / partition)
2. **groups** batches based on the leader broker
3. sends the grouped batches to the brokers
4. Pipelining if **max.in.flight.requests.per.connection** > 1



# Understand the Kafka producer

## Sender :

1. **polls** batches from the batch queues (one batch / partition)
2. **groups** batches based on the leader broker
3. sends the grouped batches to the brokers
4. Pipelining if **max.in.flight.requests.per.connection** > 1

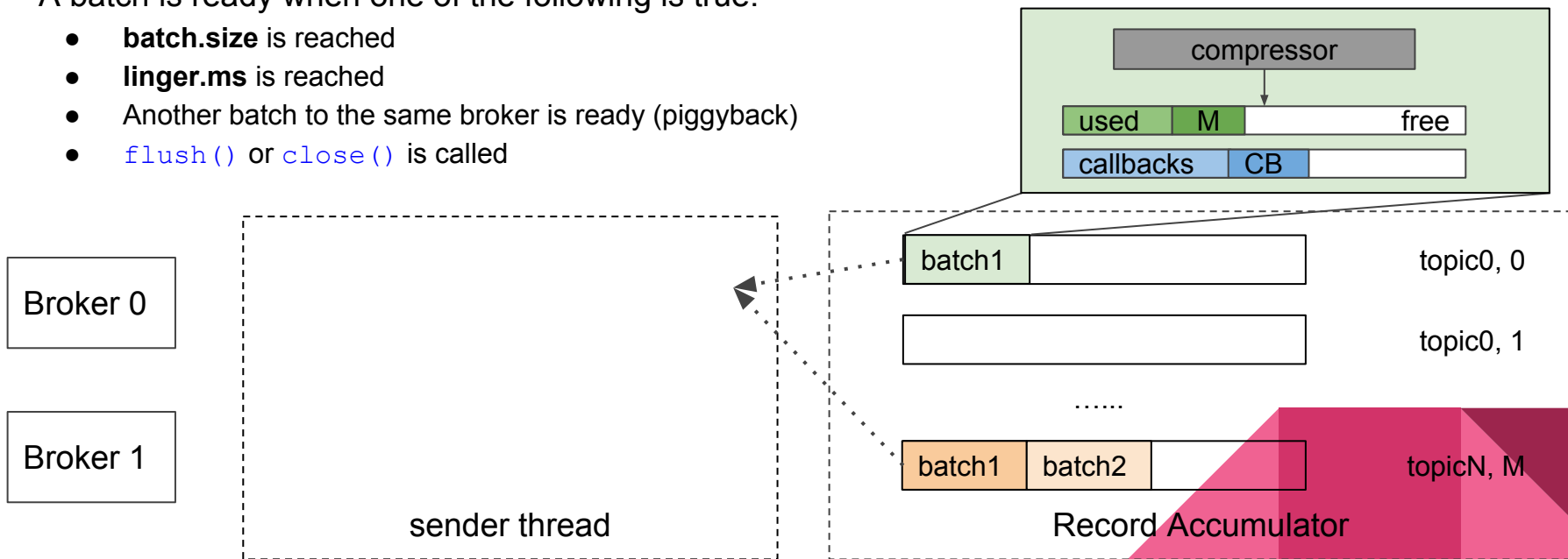


# Understand the Kafka producer

## Sender :

A batch is ready when one of the following is true:

- **batch.size** is reached
- **linger.ms** is reached
- Another batch to the same broker is ready (piggyback)
- `flush()` or `close()` is called

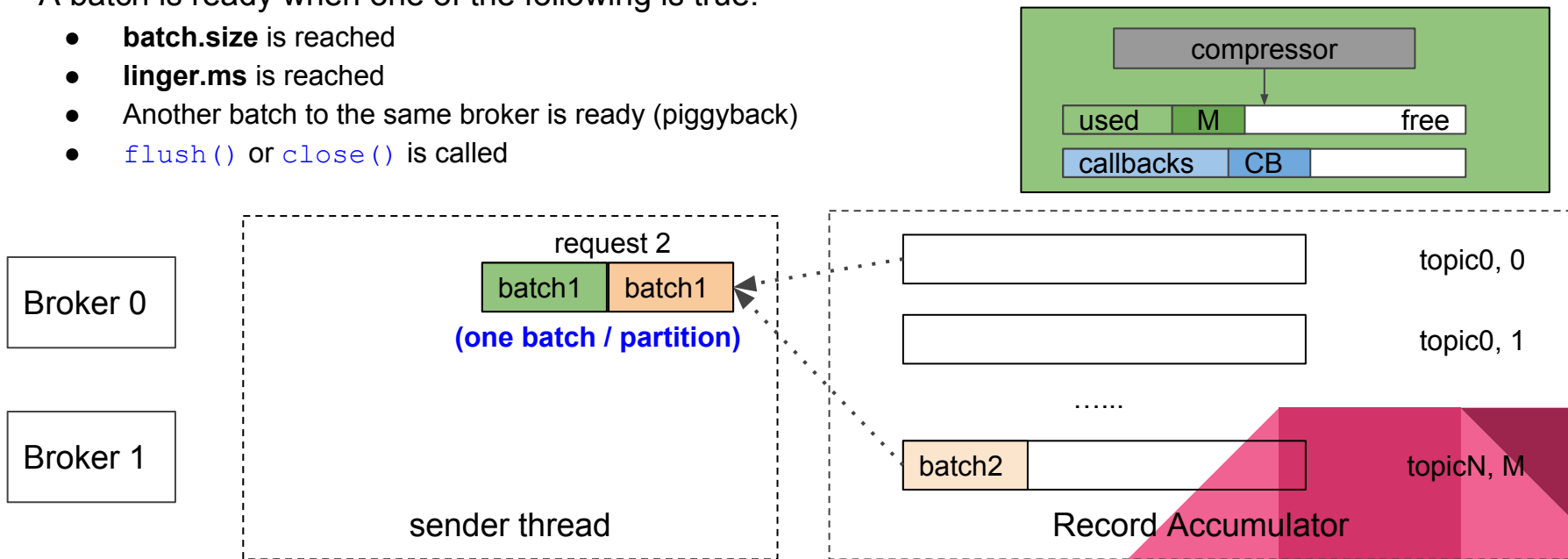


# Understand the Kafka producer

## Sender :

A batch is ready when one of the following is true:

- **batch.size** is reached
- **linger.ms** is reached
- Another batch to the same broker is ready (piggyback)
- `flush()` or `close()` is called

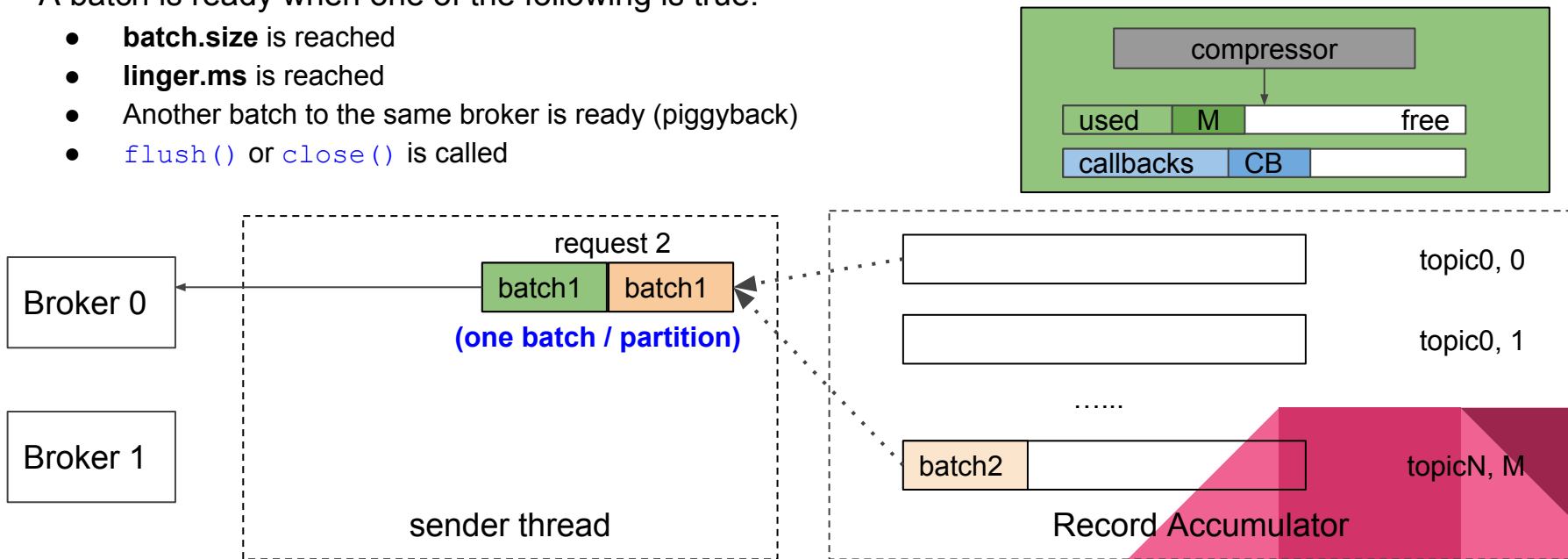


# Understand the Kafka producer

## Sender :

A batch is ready when one of the following is true:

- **batch.size** is reached
- **linger.ms** is reached
- Another batch to the same broker is ready (piggyback)
- `flush()` or `close()` is called

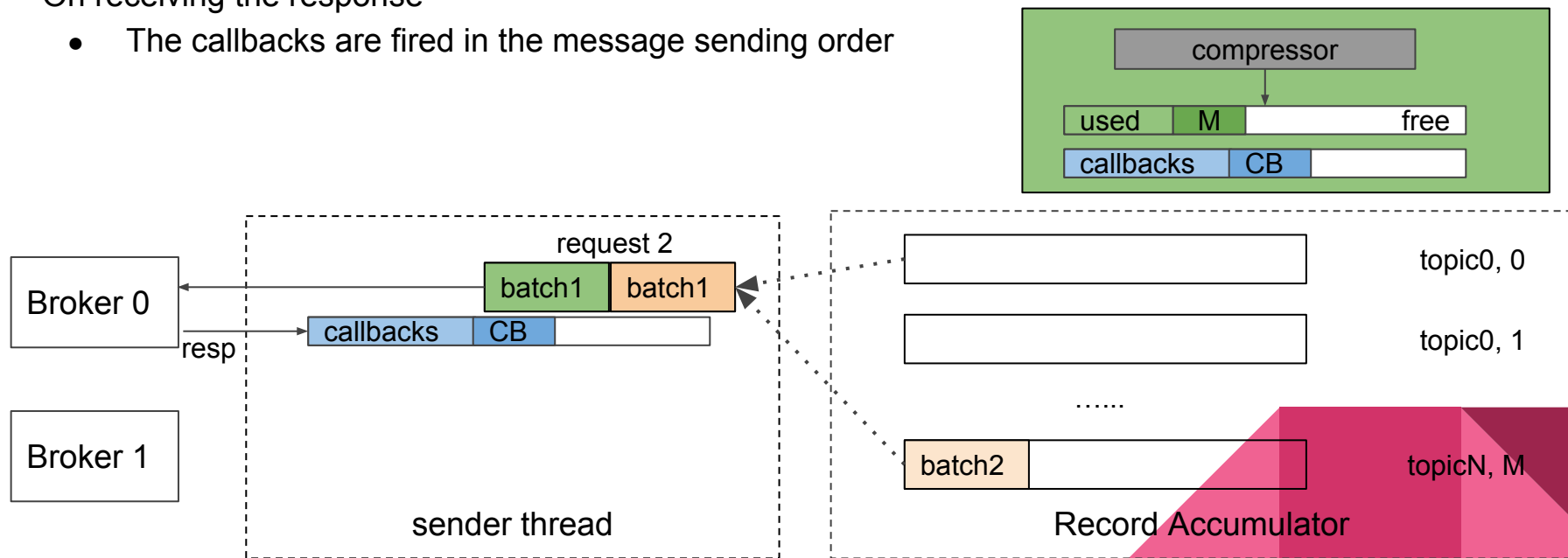


# Understand the Kafka producer

## Sender :

On receiving the response

- The callbacks are fired in the message sending order



# batch.size & linger.ms

**batch.size** is **size based** batching

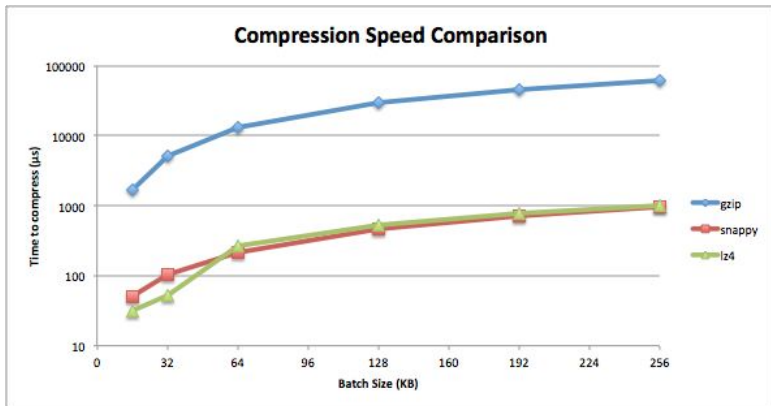
**linger.ms** is **time based** batching

In general, more batching

- $\Rightarrow$  **Better** compression ratio  $\Rightarrow$  **Higher** throughput
- $\Rightarrow$  **Higher** latency

# compression.type

- Compression is usually the **dominant** part of the `producer.send()`
- The speed of different compression types differs **A LOT**
- Compression is in user thread, so **adding more user threads** helps with **throughput** if compression is slow





# acks

Defines different durability level for producing.

acks	Throughput	Latency	Durability
0	high	low	No guarantee
1	medium	medium	leader
-1	low	high	ISR

# max.in.flight.requests.per.connection

max.in.flight.requests.per.connection > 1 means pipelining.

In general, pipelining

- gives **better throughput**
- may cause **out of order** delivery when retry occurs
- **Excessive pipelining** ⇒ Drop of throughput
  - lock contention
  - worse batching

We will be using **max.in.flight.requests.per.connection=1** for today's talk.

# Agenda

- The goal of producer performance tuning
- Understand KafkaProducer
- **Producer performance tuning**
  - ProducerPerformance tool
  - Quantitative analysis using producer metrics
  - Play with a toy example
- Some real world examples
  - Latency when acks=-1
  - Produce when RTT is long
- Q & A

# ProducerPerformance Tool

Useful producer performance tuning tool:

- `org.apache.kafka.tools.ProducerPerformance`

Example:

```
./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic  
becket_test_3_replicas_1_partition --throughput 1000000 --producer-props bootstrap.  
servers=localhost:9092 max.in.flight.requests.per.connection=1 batch.size=100000  
compression.type=lz4
```

# ProducerPerformance Tool

Useful producer performance tuning tool:

- `org.apache.kafka.tools.ProducerPerformance`

With KAFKA-3554:

```
./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic  
becket_test_3_replicas_1_partition --throughput 1000000 --num-threads 2 --value-bound 50000  
--producer-props bootstrap.servers=localhost:9092 max.in.flight.requests.per.connection=1  
batch.size=100000 compression.type=lz4
```

- **--num-threads** : The number of the user threads to send messages.
- **--value-bound** : The range of the random integer in the messages. This option is useful when compression is used. Different integer range simulates different compression ratio.

# Quantitative analysis using producer metrics

The improved `ProducerPerformance` tool ([KAFKA-3554](#)) prints the following producer metrics.

- **Select\_Rate\_Avg** (The rate that the sender thread runs to check if it can send some messages)
- **Request\_Rate\_Avg**
- **Request\_Latency\_Avg** (Not including the callback execution time)
- **Request\_Size\_Avg** (After compression)
- **Batch\_Size\_Avg** (After compression)
- **Records\_Per\_Request\_Avg**
- **Record\_Queue\_Time\_Avg**
- **Compression\_Rate\_Avg**

Note: The metrics need some time (~ 1 min) to become stable.

# Quantitative analysis using producer metrics

```
./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic  
becket_test_3_replicas_4_partition --throughput 100000 --num-threads 1 --value-bound 50000  
--producer-props bootstrap.servers=localhost:9092 compression.type=gzip max.in.flight.  
requests.per.connection=1
```

1000000 records sent, 10444.300545 records/sec (9.96 MB/sec), 4.19 ms avg latency, 154.00 ms max latency, 4 ms 50th,  
6 ms 95th, 6 ms 99th, 22 ms 99.9th.

Select\_Rate\_Avg: 3114.67  
Request\_Rate\_Avg: 1448.53  
Request\_Latency\_Avg: 2.73  
Request\_Size\_Avg: 5034.47  
Batch\_Size\_Avg: 4941.93  
Records\_Per\_Request\_Avg: 7.05  
Record\_Queue\_Time\_Avg: 2.45  
Compression\_Rate\_Avg: 0.68

# Quantitative analysis using producer metrics

1.  $\text{throughput\_Avg} \approx \text{Request\_Rate\_Avg} * \text{Request\_Size\_Avg} / \text{Compression\_Rate\_Avg}$

./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic becket\_test\_3\_replicas\_4\_partition --throughput 100000 --num-threads 1 --value-bound 50000 --producer-props bootstrap.servers=localhost:9092 compression.type=gzip max.in.flight.requests.per.connection=1  
1000000 records sent, 10444.300545 records/sec (9.96 MB/sec), 4.19 ms avg latency, 154.00 ms max latency, 4 ms 50th, 6 ms 95th, 6 ms 99th, 22 ms 99.9th.

Select\_Rate\_Avg: 3114.67

**Request\_Rate\_Avg: 1448.53**

Request\_Latency\_Avg: 2.73

**Request\_Size\_Avg: 5034.47**

Batch\_Size\_Avg: 4941.93

Records\_Per\_Request\_Avg: 7.05

Record\_Queue\_Time\_Avg: 2.45

**Compression\_Rate\_Avg: 0.68**

**1448.53 \* 5034.47 / 0.68 = 10.22 MB/sec**  
(The gap is due to the request\_overhead)



# Quantitative analysis using producer metrics

2.  $\text{Request\_Size\_Avg} = \text{Records\_Per\_Request\_Avg} * \text{Record\_Size} * \text{Compression\_Rate\_Avg} + \text{Request\_Overhead}$

`./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic becket_test_3_replicas_4_partition --throughput 100000 --num-threads 1 --value-bound 50000 --producer-props bootstrap.servers=localhost:9092 compression.type=gzip max.in.flight.requests.per.connection=1`  
1000000 records sent, 10444.300545 records/sec (9.96 MB/sec), 4.19 ms avg latency, 154.00 ms max latency, 4 ms 50th, 6 ms 95th, 6 ms 99th, 22 ms 99.9th.

Select\_Rate\_Avg: 3114.67

Request\_Rate\_Avg: 1448.53

Request\_Latency\_Avg: 2.73

**Request\_Size\_Avg: 5034.47**

Batch\_Size\_Avg: 4941.93

**Records\_Per\_Request\_Avg: 7.05**

Record\_Queue\_Time\_Avg: 2.45

**Compression\_Rate\_Avg: 0.68**

$$7.05 * 1000 * 0.68 + \text{Request\_Overhead} = 4794 + \text{Request\_Overhead}$$

The **Request\_Overhead**:

- depends on the **number of topics** and **partitions**
- usually ranges from dozens bytes to hundreds of bytes.

# Quantitative analysis using producer metrics

## 3. $\text{Request\_Rate\_Upper\_Limit} = (1000 / \text{Request\_Latency\_Avg}) * \text{Num\_Brokers}$

./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic becket\_test\_3\_replicas\_4\_partition --throughput 100000 --num-threads 1 --value-bound 50000 --producer-props bootstrap.servers=localhost:9092 compression.type=gzip max.in.flight.requests.per.connection=1

1000000 records sent, 10444.300545 records/sec (9.96 MB/sec), 4.19 ms avg latency, 154.00 ms max latency, 4 ms 50th, 6 ms 95th, 6 ms 99th, 22 ms 99.9th.

Select\_Rate\_Avg: 3114.67

**Request\_Rate\_Avg: 1448.53**

**Request\_Latency\_Avg: 2.73**

Request\_Size\_Avg: 5034.47

Batch\_Size\_Avg: 4941.93

Records\_Per\_Request\_Avg: 7.05

Record\_Queue\_Time\_Avg: 2.45

Compression\_Rate\_Avg: 0.68

$$(1000 / 2.73) * 5 = 1813$$

The gap is:

- due to the producer overhead,
- larger if request rate is higher

# Quantitative analysis using producer metrics

## 4. $\text{latency\_avg} \approx (\text{Record\_Queue\_Time\_Avg} / 2) + \text{Request\_Latency\_Avg} + \text{Callback\_Latency}$

./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic becket\_test\_3\_replicas\_4\_partition --throughput 100000 --num-threads 1 --value-bound 50000 --producer-props bootstrap.servers=localhost:9092 compression.type=gzip max.in.flight.requests.per.connection=1  
1000000 records sent, 10444.300545 records/sec (9.96 MB/sec), **4.19 ms avg latency**, 154.00 ms max latency, 4 ms 50th, 6 ms 95th, 6 ms 99th, 22 ms 99.9th.

Select\_Rate\_Avg: 3114.67

Request\_Rate\_Avg: 1448.53

**Request\_Latency\_Avg: 2.73**

Request\_Size\_Avg: 5034.47

Batch\_Size\_Avg: 4941.93

Records\_Per\_Request\_Avg: 7.05

**Record\_Queue\_Time\_Avg: 2.45**

Compression\_Rate\_Avg: 0.68

$$2.45 / 2 + 2.73 + \text{Callback\_Latency} = 3.96 + \text{Callback\_Latency}$$

(The callback latency and some other time cost in the `ProducerPerformance` tool is usually small enough to ignore)

# Play with a toy example

```
./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic becket_test_3_replicas_4_partition --  
throughput 100000 --num-threads 1 --value-bound 50000 --producer-props bootstrap.servers=localhost:9092 compression.  
type=gzip max.in.flight.requests.per.connection=1
```

1000000 records sent, 10444.300545 records/sec (9.96 MB/sec), 4.19 ms avg latency, 154.00 ms max latency, 4 ms 50th,  
6 ms 95th, 6 ms 99th, 22 ms 99.9th.

Select\_Rate\_Avg: 3114.67  
Request\_Rate\_Avg: 1448.53  
Request\_Latency\_Avg: 2.73  
Request\_Size\_Avg: 5034.47  
Batch\_Size\_Avg: 4941.93  
Records\_Per\_Request\_Avg: 7.05  
Record\_Queue\_Time\_Avg: 2.45  
Compression\_Rate\_Avg: 0.68

- RTT=1.55 ms
- 5 broker cluster

# Play with a toy example

```
./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic becket_test_3_replicas_4_partition --  
throughput 100000 --num-threads 1 --value-bound 50000 --producer-props bootstrap.servers=localhost:9092 compression.  
type=gzip max.in.flight.requests.per.connection=1
```

1000000 records sent, 10444.300545 records/sec (**9.96 MB/sec**), 4.19 ms avg latency, 154.00 ms max latency, 4 ms 50th,  
6 ms 95th, 6 ms 99th, 22 ms 99.9th.

Select\_Rate\_Avg: 3114.67  
Request\_Rate\_Avg: 1448.53  
Request\_Latency\_Avg: 2.73  
Request\_Size\_Avg: 5034.47  
Batch\_Size\_Avg: 4941.93  
Records\_Per\_Request\_Avg: 7.05  
Record\_Queue\_Time\_Avg: 2.45  
Compression\_Rate\_Avg: 0.68

- Network Bandwidth = **1 Gbps**:
  - Throughput can be improved

# Play with a toy example

```
./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic becket_test_3_replicas_4_partition --  
throughput 100000 --num-threads 1 --value-bound 50000 --producer-props bootstrap.servers=localhost:9092 compression.  
type=gzip max.in.flight.requests.per.connection=1
```

1000000 records sent, 10444.300545 records/sec (9.96 MB/sec), 4.19 ms avg latency, 154.00 ms max latency, 4 ms 50th,  
6 ms 95th, 6 ms 99th, 22 ms 99.9th.

Select\_Rate\_Avg: 3114.67

**Request\_Rate\_Avg: 1448.53**

Request\_Latency\_Avg: 2.73

**Request\_Size\_Avg: 5034.47**

Batch\_Size\_Avg: 4941.93

Records\_Per\_Request\_Avg: 7.05

Record\_Queue\_Time\_Avg: 2.45

**Compression\_Rate\_Avg: 0.68**

- RTT = 1.55 ms:
  - Request Latency is not bad.
- 1448 Request rate is not bad
  - 5 brokers \* (1000 / 2.73) = 1813 (theoretical upper limit)
- **Each request is too small**
  - Default batch size = 16KB

$\text{throughput\_Avg} \sim \text{Request\_Rate\_Avg} * \text{Request\_Size\_Avg} / \text{Compression\_Rate\_Avg}$

# Play with a toy example

```
./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic becket_test_3_replicas_4_partition --  
throughput 100000 --num-threads 1 --value-bound 50000 --producer-props bootstrap.servers=localhost:9092 compression.  
type=gzip max.in.flight.requests.per.connection=1  
1000000 records sent, 10444.300545 records/sec (9.96 MB/sec), 4.19 ms avg latency, 154.00 ms max latency, 4 ms 50th,  
6 ms 95th, 6 ms 99th, 22 ms 99.9th.
```

Select\_Rate\_Avg: 3114.67

**Request\_Rate\_Avg: 1448.53**

Request\_Latency\_Avg: 2.73

**Request\_Size\_Avg: 5034.47**

Batch\_Size\_Avg: 4941.93

Records\_Per\_Request\_Avg: 7.05

Record\_Queue\_Time\_Avg: 2.45

**Compression\_Rate\_Avg: 0.68**

## Ways to increase request size:

1. Add more user threads
2. Increase number of partitions
3. Increase linger.ms (more batching)

$\text{throughput\_Avg} \sim \text{Request\_Rate\_Avg} * \text{Request\_Size\_Avg} / \text{Compression\_Rate\_Avg}$

# Play with a toy example

```
./kafka-producer-perf-test.sh --num-records 1000000 --record-size 1000 --topic becket_test_3_replicas_4_partition --  
throughput 100000 --num-threads 1 --value-bound 50000 --producer-props bootstrap.servers=localhost:9092 compression.  
type=gzip max.in.flight.requests.per.connection=1
```

1000000 records sent, 10444.300545 records/sec (**9.96 MB/sec**), 4.19 ms avg latency, 154.00 ms max latency, 4 ms 50th,  
6 ms 95th, 6 ms 99th, 22 ms 99.9th.

Select\_Rate\_Avg: 3114.67

**Request\_Rate\_Avg: 1448.53**

Request\_Latency\_Avg: 2.73

**Request\_Size\_Avg: 5034.47**

Batch\_Size\_Avg: 4941.93

Records\_Per\_Request\_Avg: 7.05

Record\_Queue\_Time\_Avg: 2.45

**Compression\_Rate\_Avg: 0.68**

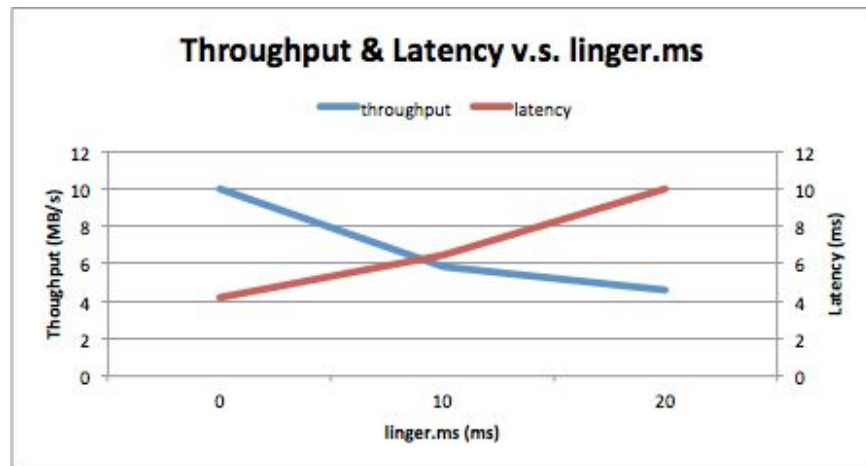
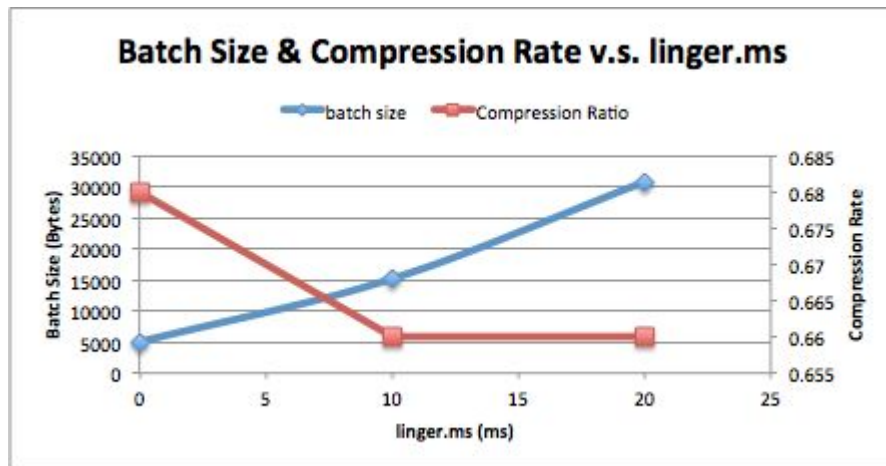
## Ways to increase request size:

1. Add more user threads
2. Increase number of partitions
3. Increase **linger.ms** (more batching)?

**throughput\_Avg** ~= **Request\_Rate\_Avg \* Request\_Size\_Avg / Compression\_Rate\_Avg**

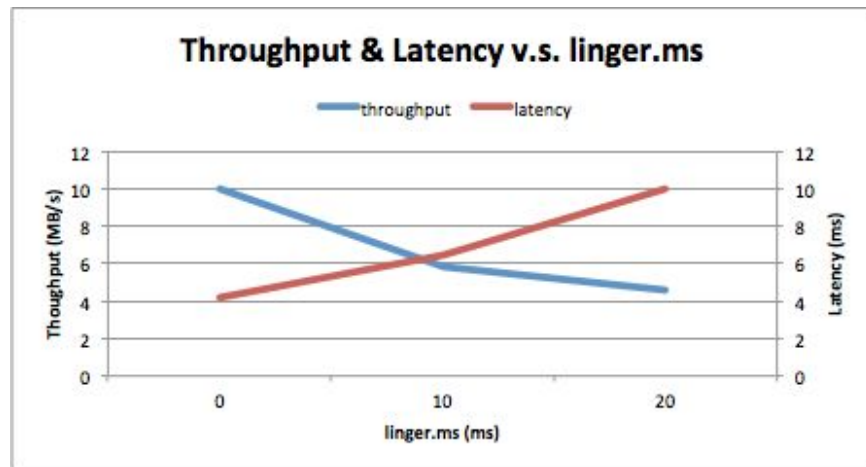
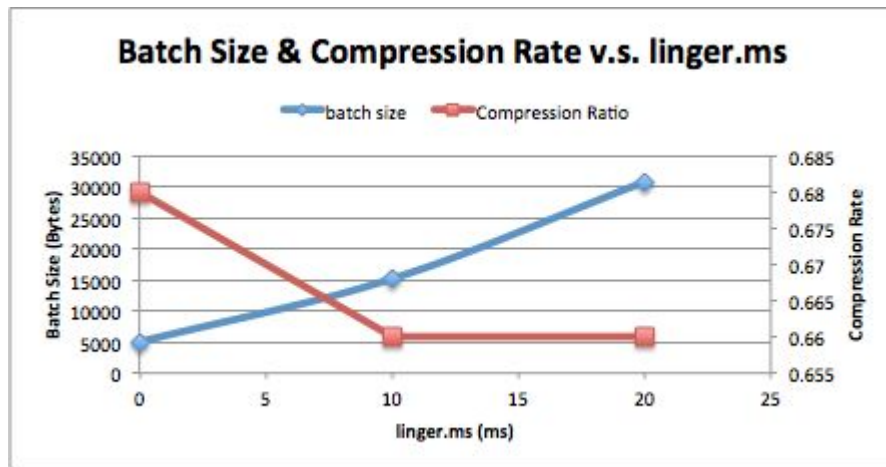


# More Batching



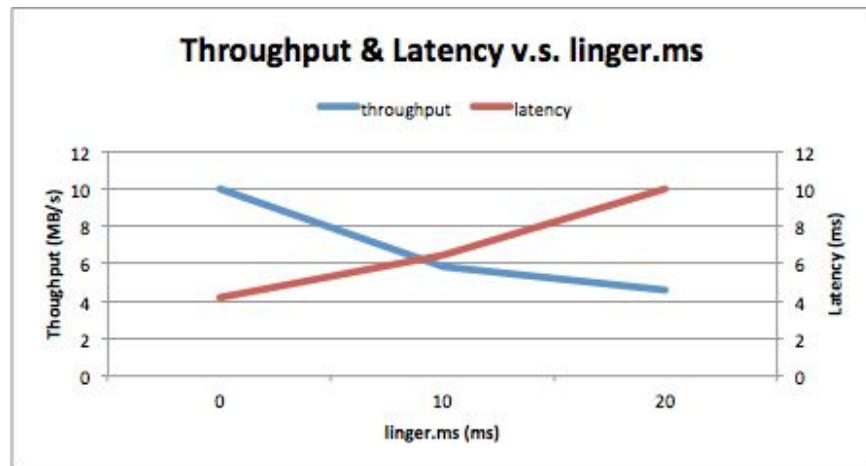
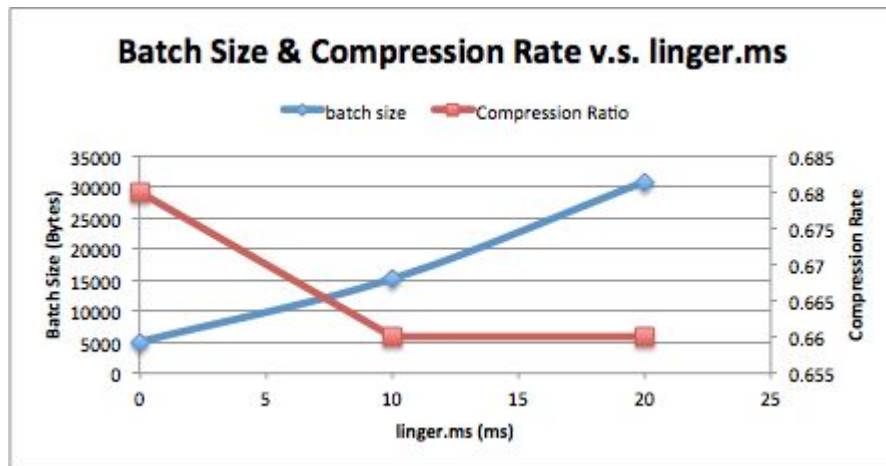
More batching does not help!?

# More Batching



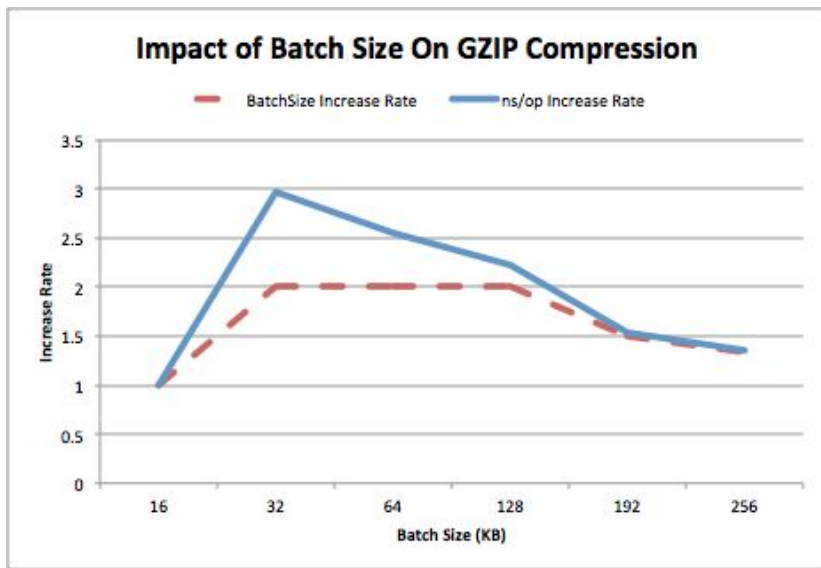
More batching does not help!  
Compression Ratio is not improved much!

# More Batching



More batching does not help!  
**Compression Ratio is not improved much!**  
But why throughput becomes worse!?

# Impact of Batch Size on GZIP Compression

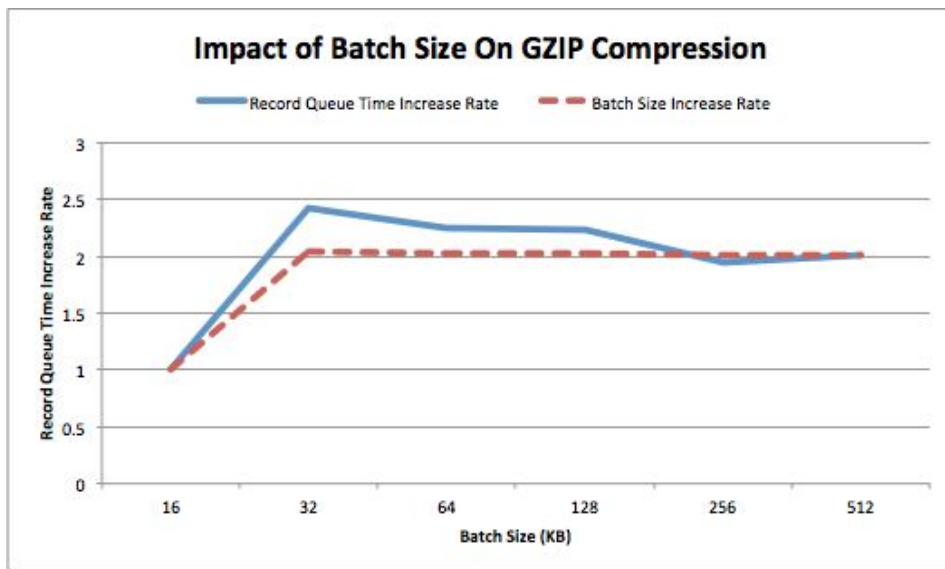


batch size (KB)	Time to fill up the batch (ns)
16	1768583.939
32	5234432.44
64	13307411.93
128	29539258.9
192	45549678.03
256	61579312.57

When batch size is **doubled from 16K to 32K**, the time needed to fill in the batch is almost **tripled**!

(The results are from JMH Micro Benchmark)

# Impact of Batch Size on GZIP Compression

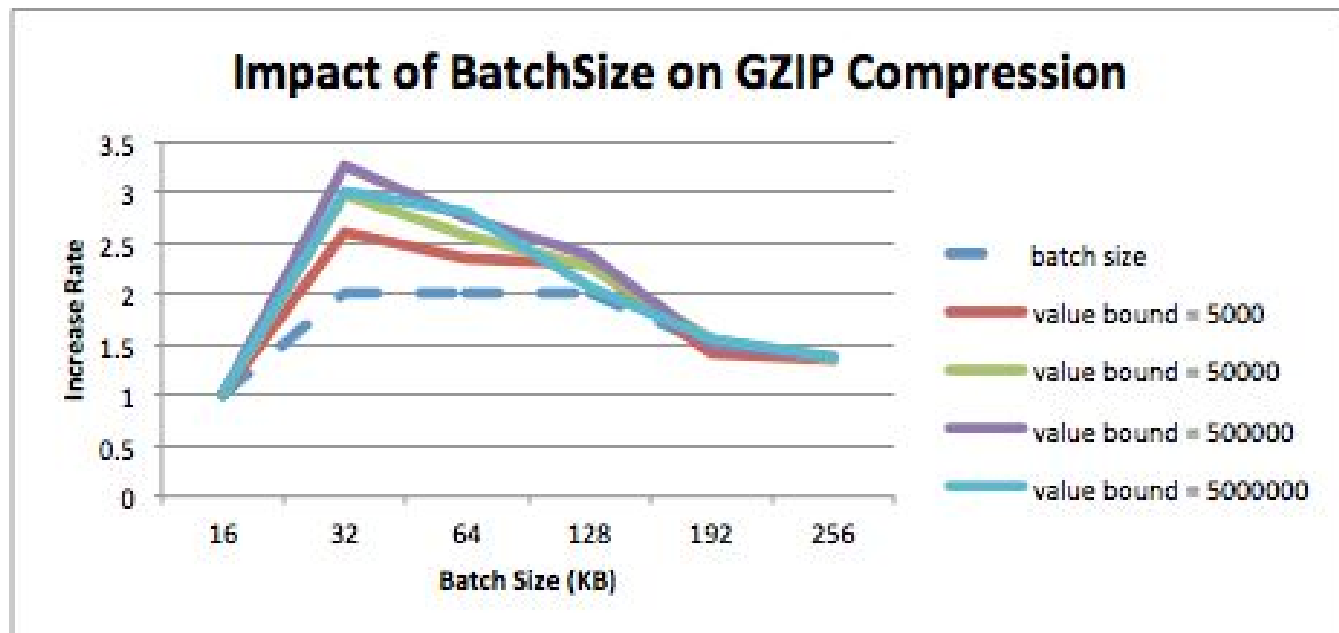


Batch_Size_Avg:	Record_Queue_Time_Avg:
15075.84	3.86
30681.25	9.37
61687.88	21.12
124451.51	47.2
248860.43	92.14
498640.58	184.84

When batch size is **doubled from 16K to 32K**, the time needed to fill in the batch is **2.5x!**

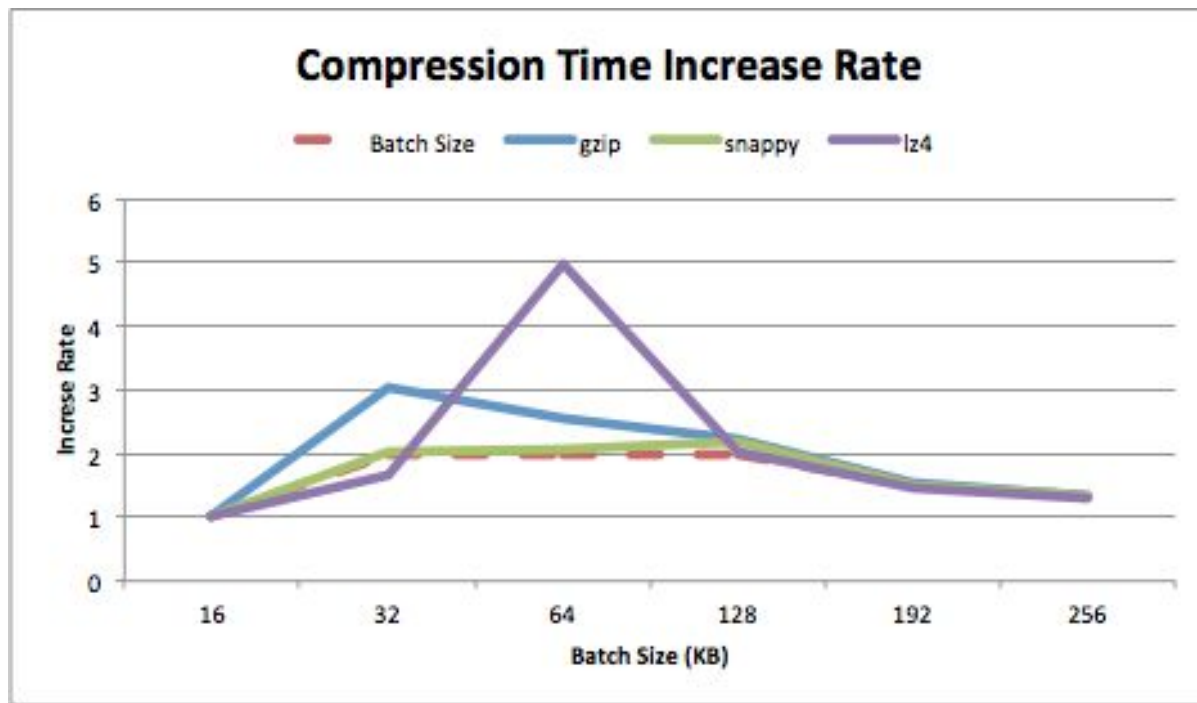
(The results are from ProducerPerformance tool with 1 user thread, 1 partition, linger.ms=20000)

# Impact of Batch Size on GZIP Compression



Different value bounds show similar results.

# Impact of Batch Size on Compression



Snappy and LZ4 has similar impact at different batch size.

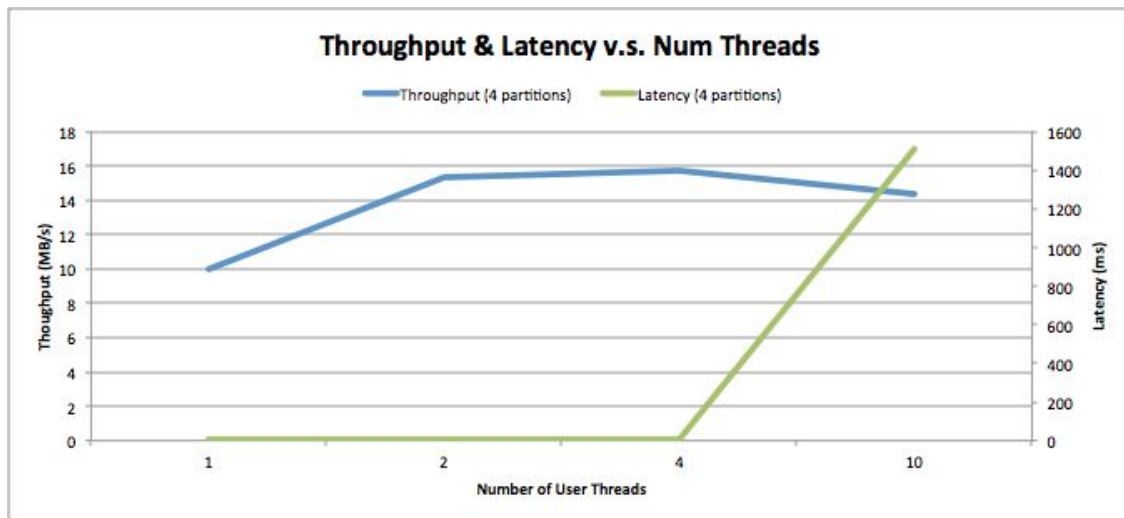
# When to increase batching

- Compression Ratio Gain > Compression Speed Loss
- Bottleneck is not in the user threads



# Play with a toy example

**OK... Let's increase number of user threads then.**

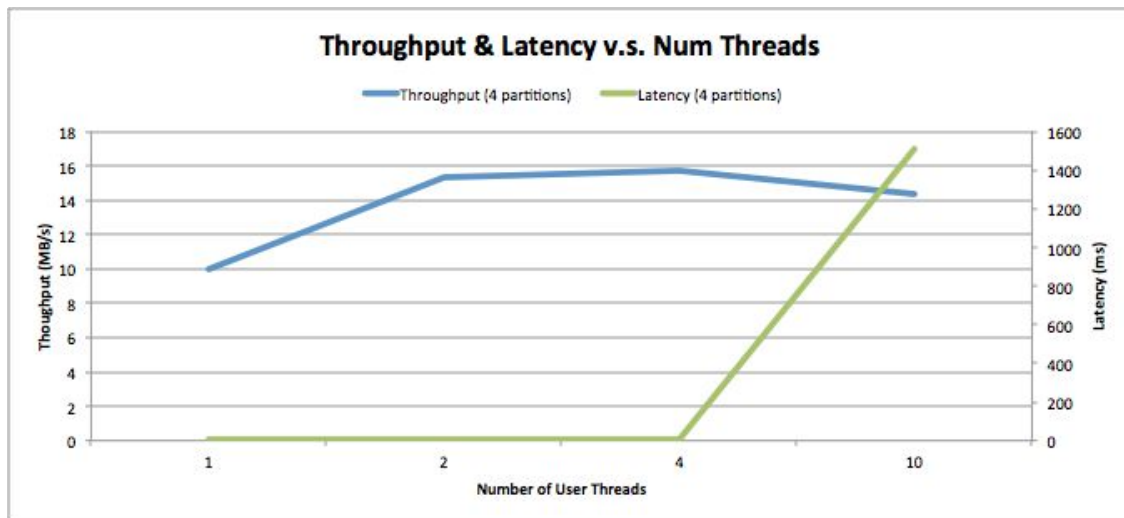


When there are 10 user threads:

- Throughput drops
- Latency soars

# Play with a toy example

OK... Let's increase number of user threads then.



When there are 10 user threads:

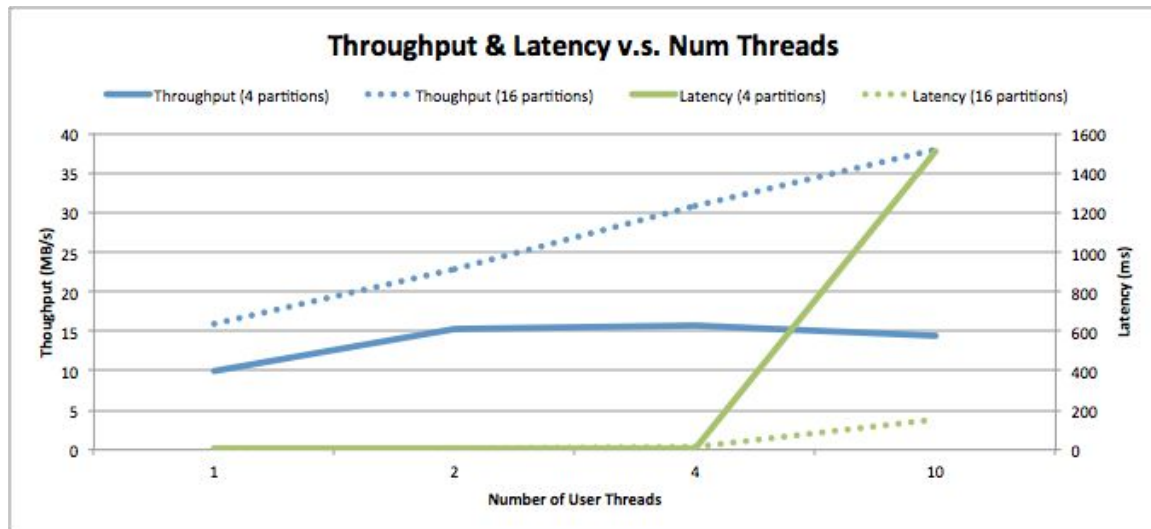
- Throughput drops
- Latency soars

Records Queue Time = 3323.85 ms

- Lock contention (10 user threads V.S. 4 partitions)
- batches are piling up!

# Play with a toy example

## When topic has 16 partitions



Batches are still piling up, but it seems much better than before.

Records Queue Time = 252.69 ms

# Play with a toy example

**Let's try tweak the batch size.**

**--num-threads 10, batch.size=16K, 16 partitions**

2000000 records sent, 39791.492579 records/sec (**37.95 MB/sec**), 152.77 ms avg latency, 1182.00 ms max latency, 14 ms 50th, 841 ms 95th, 1128 ms 99th, 1162 ms 99.9th.

**(Requests are piling up, so the bottleneck is the sender thread)**

**--num-threads 10, batch.size=800K, 16 partitions**

2000000 records sent, 35647.446752 records/sec (**34.00 MB/sec**), **20.31 ms avg latency**, 502.00 ms max latency, 17 ms 50th, 37 ms 95th, 68 ms 99th, 325 ms 99.9th.

**(Requests are no longer piling up, the bottleneck shifted to user threads)**

**Usually bigger batch leads to bigger latency.**

**But increasing batch size can improve latency by preventing the batches from piling up!**

# Play with a toy example

**To improve both latency & throughput? Increase number of partitions again.**

**--num-threads 10, batch.size=16K**

2000000 records sent, 39791.492579 records/sec (**37.95 MB/sec**), 152.77 ms avg latency, 1182.00 ms max latency, 14 ms 50th, 841 ms 95th, 1128 ms 99th, 1162 ms 99.9th.

**--num-threads 10, batch.size=800K**

2000000 records sent, 35647.446752 records/sec (**34.00 MB/sec**), 20.31 ms avg latency, 502.00 ms max latency, 17 ms 50th, 37 ms 95th, 68 ms 99th, 325 ms 99.9th.

**--num-threads 10, batch.size=16K, 32 partitions.**

2000000 records sent, 43432.939541 records/sec (**41.42 MB/sec**), **47.29 ms avg latency**, 291.00 ms max latency, 22 ms 50th, 225 ms 95th, 246 ms 99th, 270 ms 99.9th.

# Find the throughput bottleneck

## Is the bottleneck in user thread?

- Increase --num-threads and see if throughput increases accordingly
- Pay attention to lock contention

## Is the bottleneck in sender thread?

- Is throughput (MB/sec)  $\ll$  network bandwidth
- Is Record\_Queue\_Time\_Avg large?
- Is Batch\_Size\_Avg almost equals to batch.size?

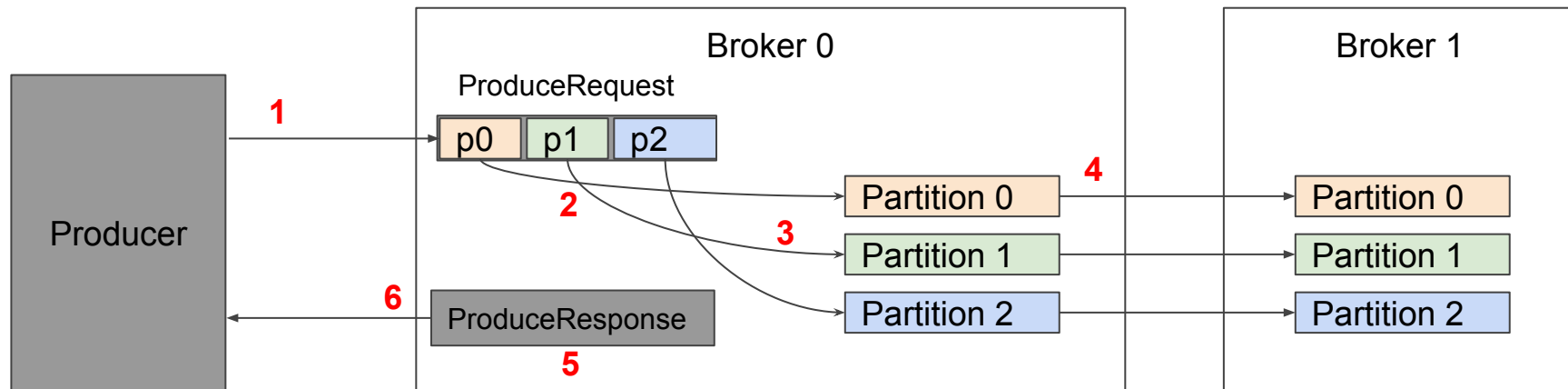
## Is the bottleneck in broker?

- Is the request latency very large?

# Agenda

- The goal of producer performance tuning
- Understand KafkaProducer
- Producer performance tuning
  - ProducerPerformance tool
  - Quantitative analysis using producer metrics
  - Play with a toy example
- **Some real world examples**
  - **Latency when acks=-1**
  - Produce when RTT is long
- Q & A

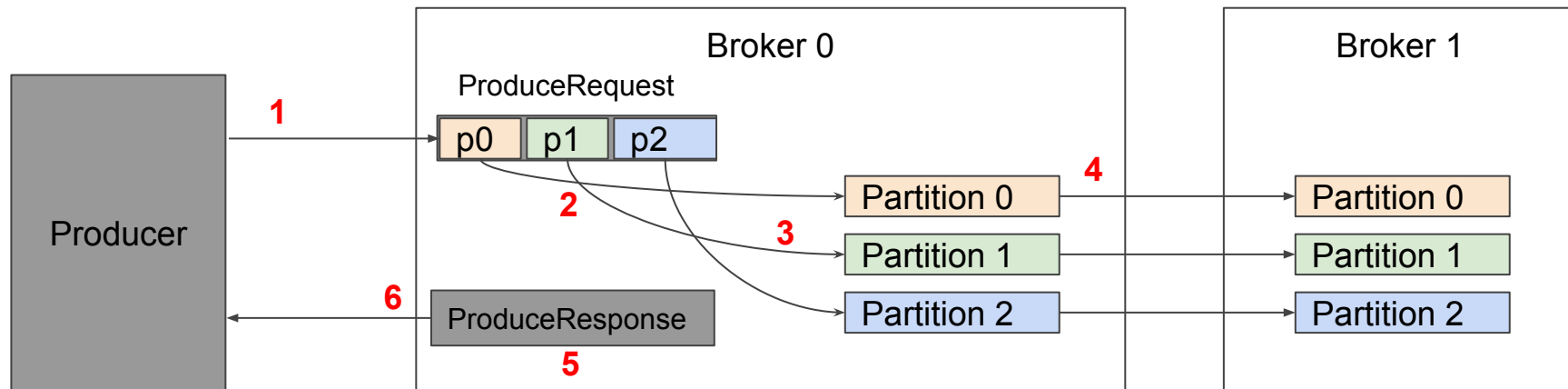
# Latency when acks=-1



1. [Network] ProduceRequest Send Time
2. [Broker] ProduceRequest Queue Time
3. [Broker] ProduceRequest Local Time
4. [Broker] ProduceRequest Remote Time
5. [Broker] ProduceResponse Queue Time
6. [Broker] ProduceResponse Send Time

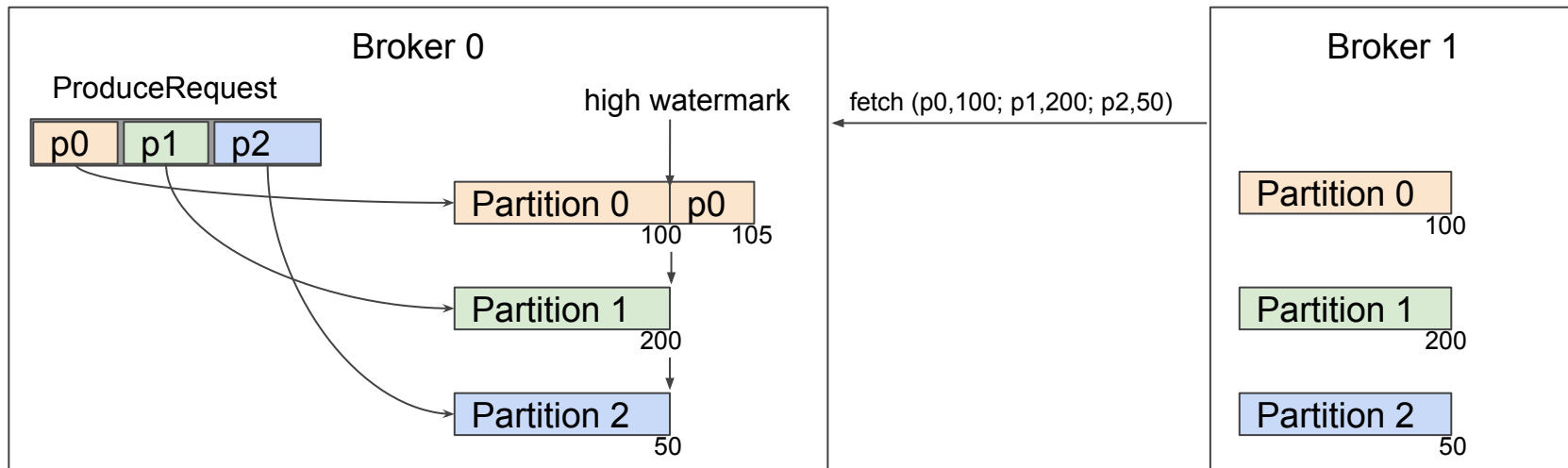


# Latency when acks=-1



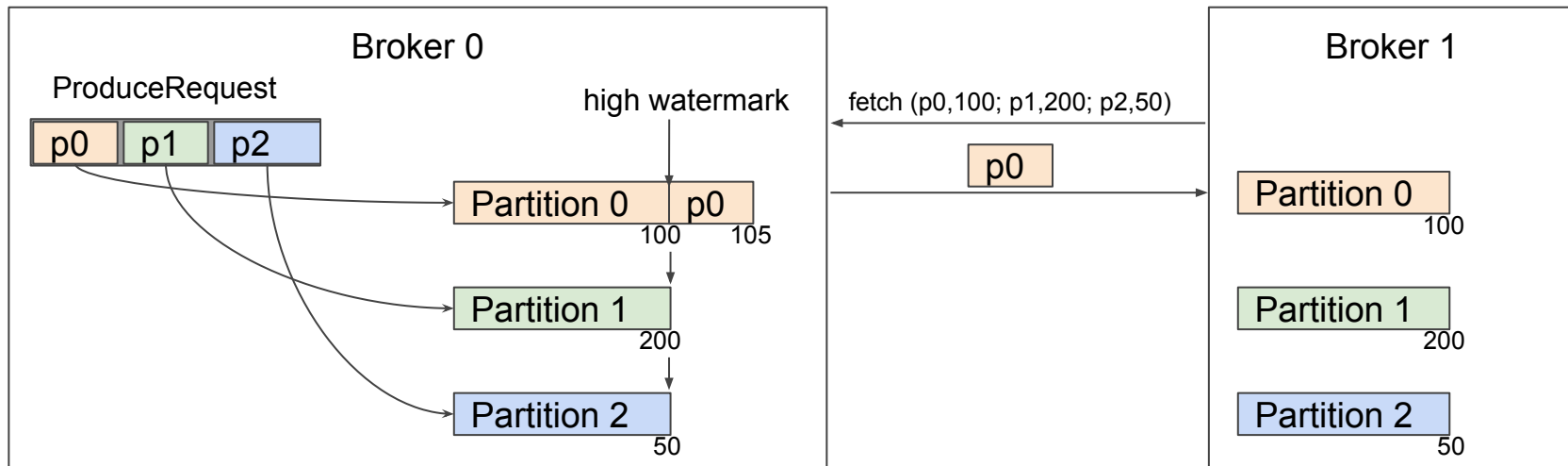
1. [Network] `ProduceRequest` Send Time
2. [Broker] `ProduceRequest` Queue Time
3. [Broker] `ProduceRequest` Local Time
4. [Broker] `ProduceRequest` Remote Time (Replication Time)
5. [Broker] `ProduceResponse` Queue Time
6. [Broker] `ProduceResponse` Send Time

# Latency when acks=-1



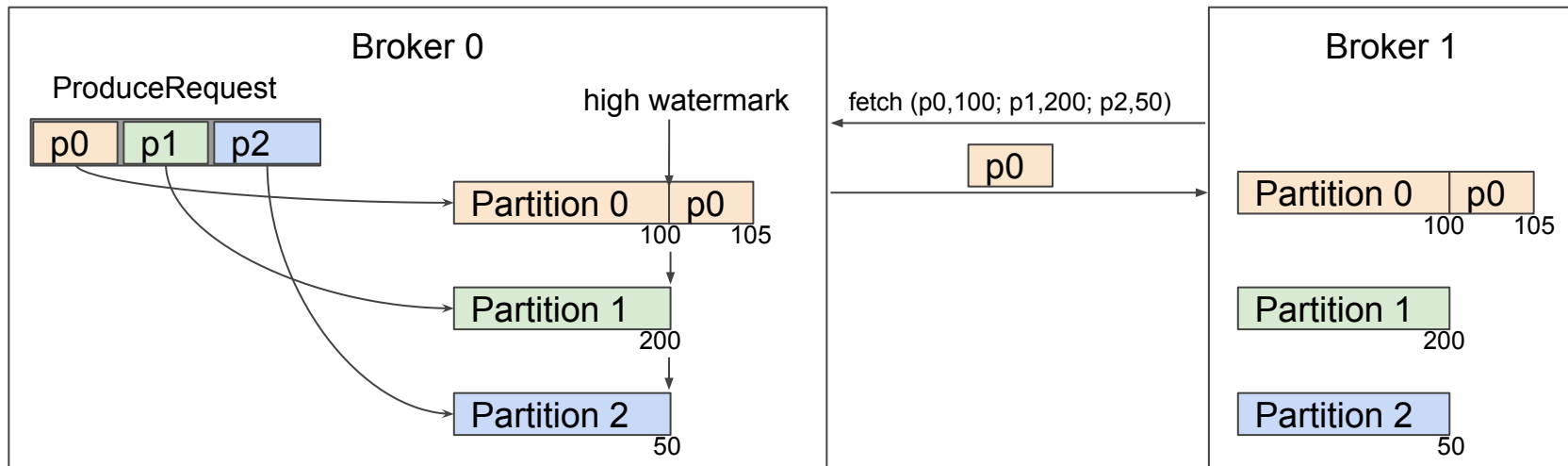
- The followers fetches data of the partitions from the leader.
- The leader uses **high watermarks** to the **In Sync Replicas (ISR)** of the all partitions

# Latency when acks=-1



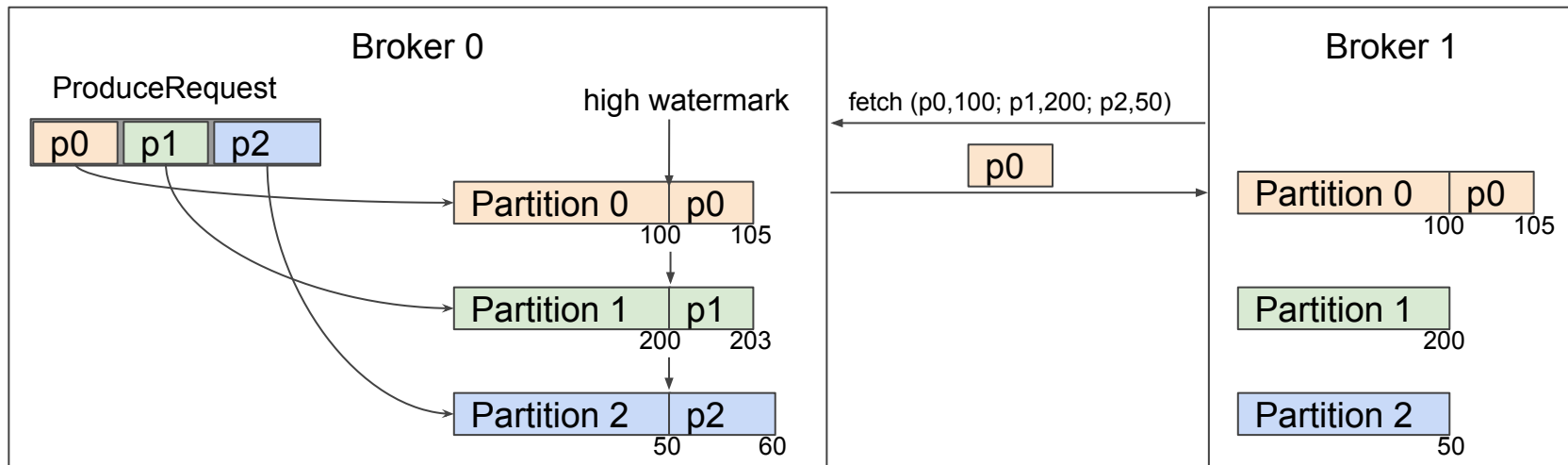
The high watermarks are not updated after the data is sent.

# Latency when acks=-1



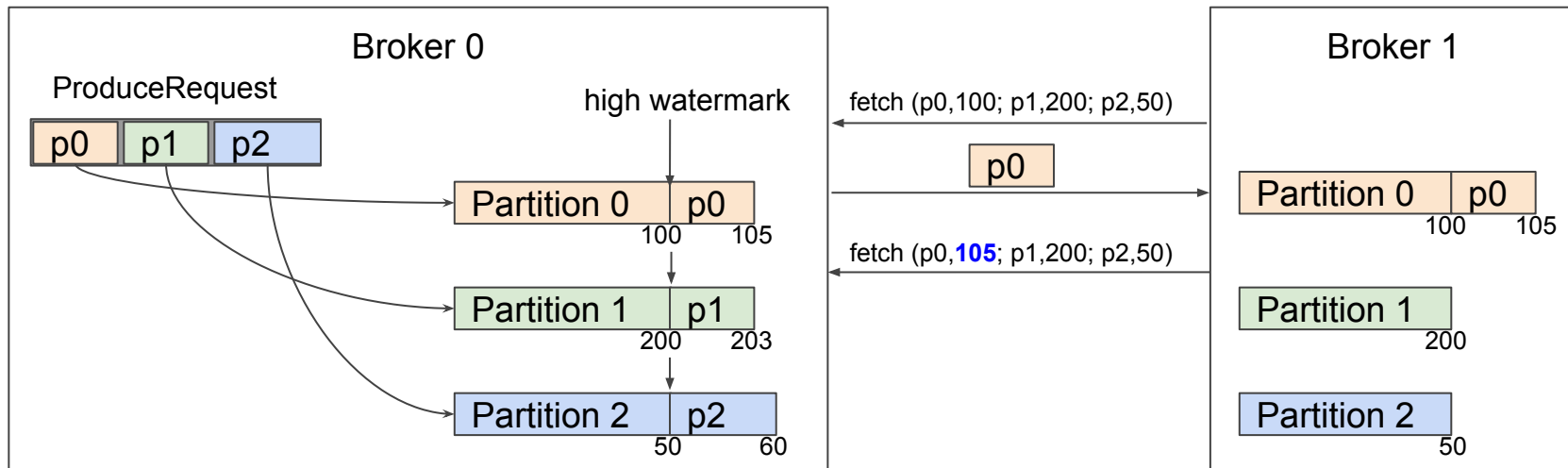
The high watermarks are not updated after the data is sent.

# Latency when acks=-1



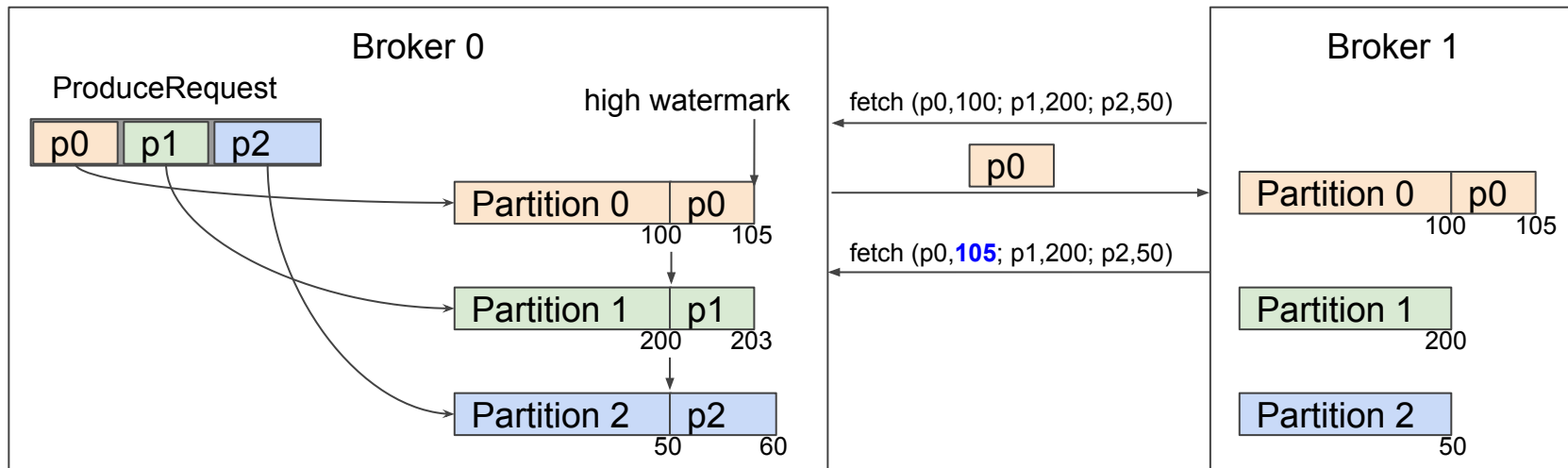
The high watermarks are not updated after the data is sent.

# Latency when acks=-1



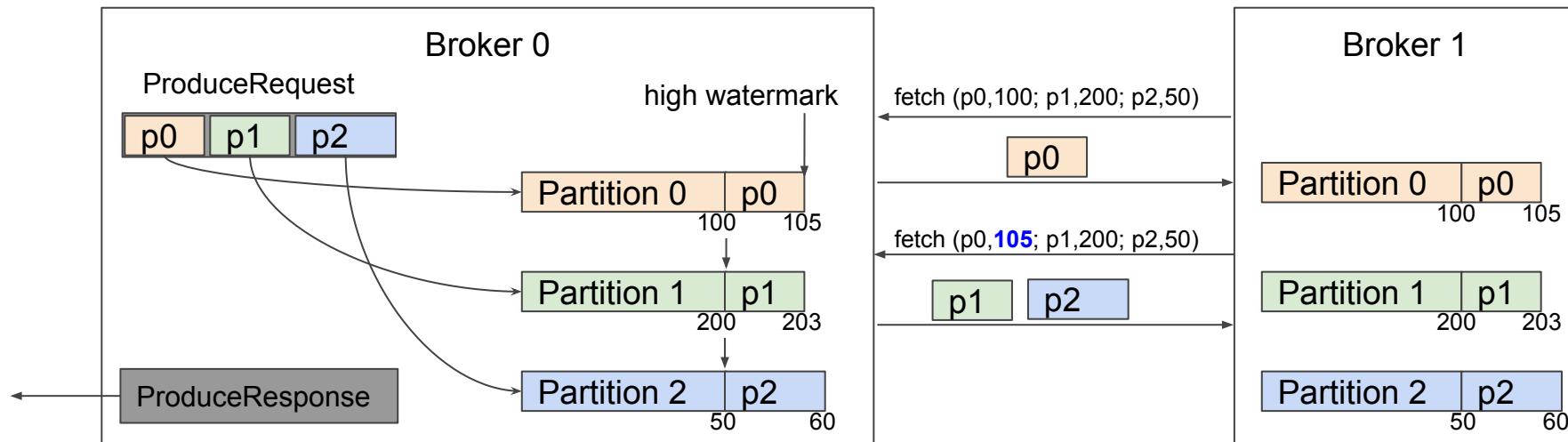
The high watermark increases **lazily**. There is one fetch delay.

# Latency when acks=-1



The high watermark increases **lazily**. There is one fetch delay.

# Latency when acks=-1

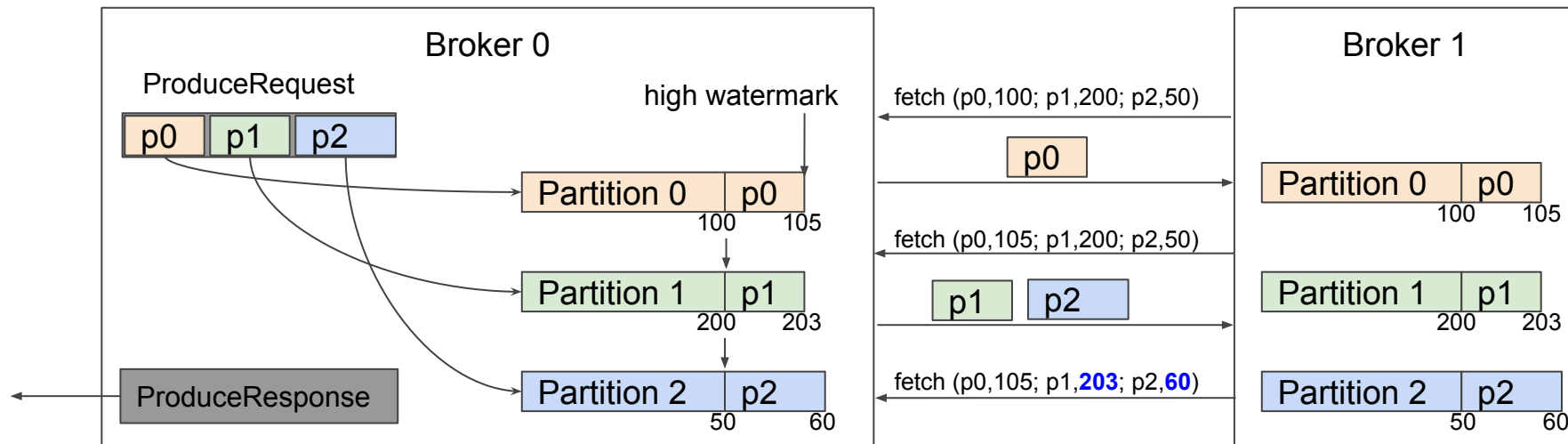


**The replication is not `ProduceRequest` aware.**

- The messages produced in the same `ProduceRequest` may be replicated in multiple fetches.
- `ProduceRequests` interfere with each other.

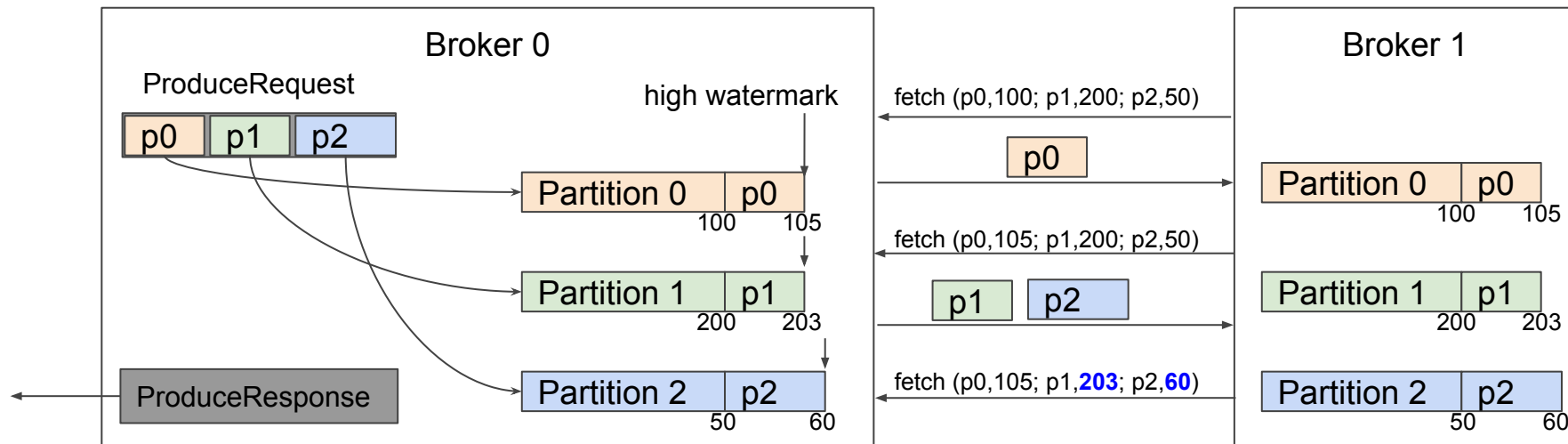


# Latency when acks=-1



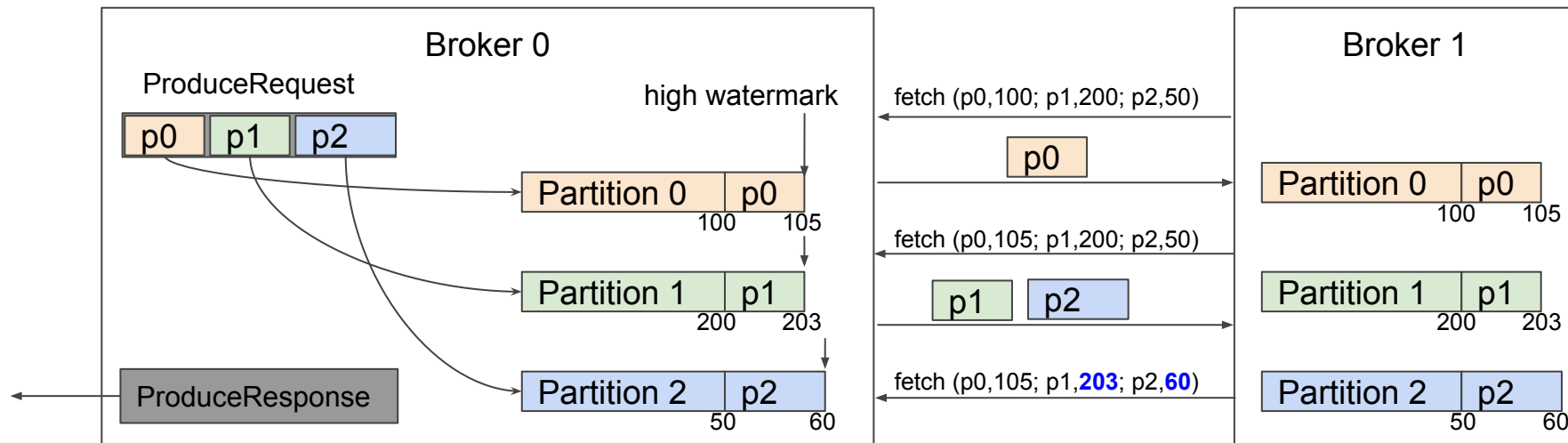
Only when the high watermarks of all the partitions has passed the offset of the appended messages in the `ProduceRequest`, the `ProduceResponse` will be sent.

# Latency when acks=-1



Only when the high watermarks of all the partitions has passed the offset of the appended messages in the ProduceRequest, the ProduceResponse will be sent.

# Latency when acks=-1



Assuming broker 1 only has one replication thread,  
Replication Time  $\sim$  Num\_Fetches \* (Local\_Time\_On\_The\_Follower +  
Fetch\_Request\_Total\_Time\_On\_The\_Leader)

# Latency when acks=-1

Replication Time  $\sim$  Num\_Fetches \* (Local\_Time\_On\_The\_Follower + Fetch\_Request\_Total\_Time\_On\_The\_Leader) / num.replica.fetchers

- Increase num.replica.fetchers
  - Parallel fetch
  - Each replica fetcher fetches from a distinct set of partitions.
- Not perfect solution
  - Diminishing effect
  - Scalability concern
    - Replica fetchers per broker = (Cluster\_Size - 1 ) \* num.replica.fetchers

# Latency when acks=-1

How many replica fetchers are enough?

- The latency target is met
- If **Replica FetchRequest Remote Time > 0**
  - The replica fetch requests are waiting for the messages to arrive.
  - Increasing num.replica.fetchers won't improve latency.
- **Partitions per replica fetcher** is low

# Agenda

- The goal of producer performance tuning
- Understand KafkaProducer
- Producer performance tuning
  - ProducerPerformance tool
  - Quantitative analysis using producer metrics
  - Play with a toy example
- **Some real world examples**
  - Latency when acks=-1
  - **Produce when RTT is long**
- Q & A

# Produce when RTT is long

We have a cross-ocean pipeline using remote produce

- RTT is big ( $\sim 200$  ms)
- Bandwidth = 1 Gb/s
- Batch size 800K
- Linger.ms 30ms
- 64 Partitions
- 8 user threads

# Produce when RTT is long

We have a cross-ocean pipeline using remote produce

- RTT is big ( $\sim 200$  ms)
- Bandwidth = 1 Gb/s
- Batch size 800K
- Linger.ms 30ms
- 64 Partitions
- 8 user threads
- **Throughput < 1 MB/s**



# Produce when RTT is long

**The TCP connection needs more send and receive buffer.**

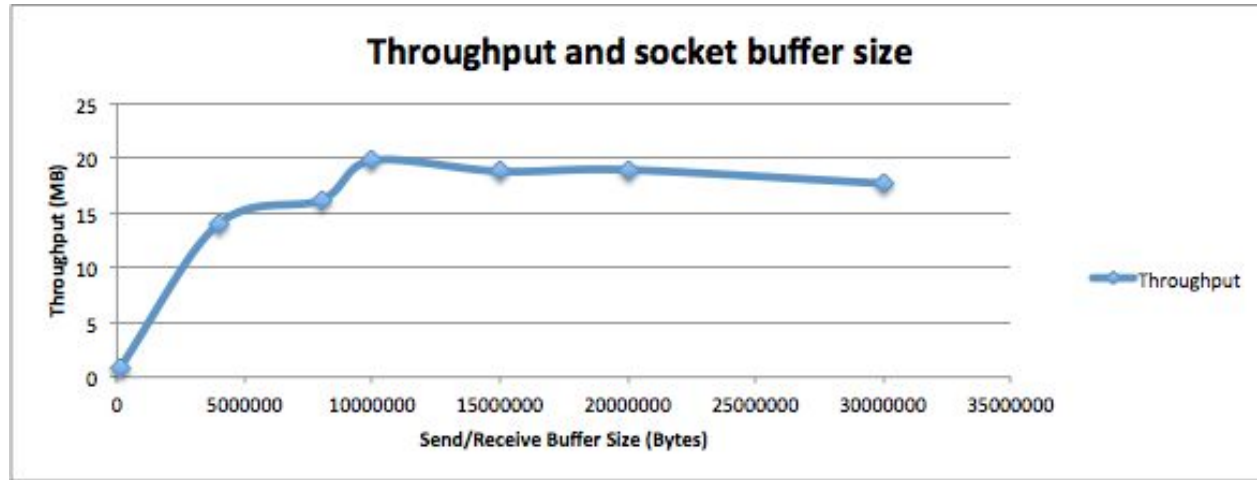
Broker default: `socket.receive.buffer.bytes` = 100K

Producer default: `send.buffer.bytes` = 128K

Theoretical throughput with default setting =  $100\text{K} / \text{RTT} = 500\text{KB}$

Theoretical best buffer size =  $\text{RTT} * \text{Bandwidth} = 0.2\text{s} * 1 \text{ Gb/s} = \mathbf{25 \text{ MB}}$

# Produce when RTT is long



Changing socket buffer size may require OS TCP buffer limit.

# Produce when RTT is long

Changing socket buffer size may require OS TCP buffer limit.

```
# The default setting of the socket receive buffer in bytes.  
net.core.rmem_default = 124928
```

```
# The maximum receive socket buffer size in bytes.  
net.core.rmem_max = 2048000
```

```
# The default setting (in bytes) of the socket send buffer.  
net.core.wmem_default = 124928
```

```
# The maximum send socket buffer size in bytes.  
net.core.wmem_max = 2048000
```

# Q&A

Backup slides are following.

# Producer Metrics With 4 Partitions

	1 thread	2 threads	4 threads	10 threads
Select_Rate_Avg:	3114.67	682.55	1359.05	2641.75
Request_Rate_Avg:	1448.53	1282.76	840.36	634.22
Request_Latency_Avg:	2.73	3.11	4.62	6.09
Request_Size_Avg:	5034.47	8703.1	13192.99	14931.51
Batch_Size_Avg:	4941.93	8607.87	13098.74	14837.54
Records_Per_Request_Avg:	7.05	12.46	19.08	21.63
Record_Queue_Time_Avg:	2.45	2.79	5.17	3323.85
Compression_Rate_Avg:	0.68	0.67	0.66	0.66

# Producer Metrics With 16 Partitions

	1 Thread	2 Threads	4 Threads	10 Threads
Select_Rate_Avg:	2746.74	1948.27	1241.82	2199.14
Request_Rate_Avg:	1270.49	1093.21	812.54	568.09
Request_Latency_Avg:	3.91	4.53	6.07	8.16
Request_Size_Avg:	8758.02	15447.04	27688.3	45652.14
Batch_Size_Avg:	2706.87	4808.15	8638.38	14283.30
Records_Per_Request_Avg:	12.01	21.82	39.86	66.33
Record_Queue_Time_Avg:	3.13	3.86	6.42	252.69
Compression_Rate_Avg:	0.68	0.67	0.66	0.66

# Lock contention for sender thread

Threads	2	4	8	10
linger.ms	50	30	20	15
Record_Queue_Time_Avg	50	31.08	21.21	17.27
Difference	0	1.08	1.21	2.27

This table shows some evidence of lock contention in the Kafka producer.

For a topic with 16 partitions, we do the following:

- Set linger.ms to make them almost the generating the similar size of batch.
- Compare the difference between linger.ms and Record\_Queue\_Time\_Avg
- The bigger the difference is, the worse the lock contention is between the sender thread and the user threads.