HITACHI
Inspire the Next

めざせ！Kafkaマスター
～Apache Kafkaで最高の性能を出すには～

2017年12月8日

株式会社日立製作所　OSSソリューションセンタ
伊藤雅博

自己紹介

HITACHI
Inspire the Next

- 伊藤 雅博 (いとう まさひろ)
  - 所属： 株式会社日立製作所　OSSソリューションセンタ
  - 業務： Hadoop/Sparkを中心としたビッグデータ関連OSSの導入支援やテクニカルサポート

目次

HITACHI
Inspire the Next

1. Apache Kafkaとは
2. Kafkaの内部構造とチューニングポイント
3. 性能検証の概要
4. 検証結果と考察
5. まとめ

HITACHI
Inspire the Next

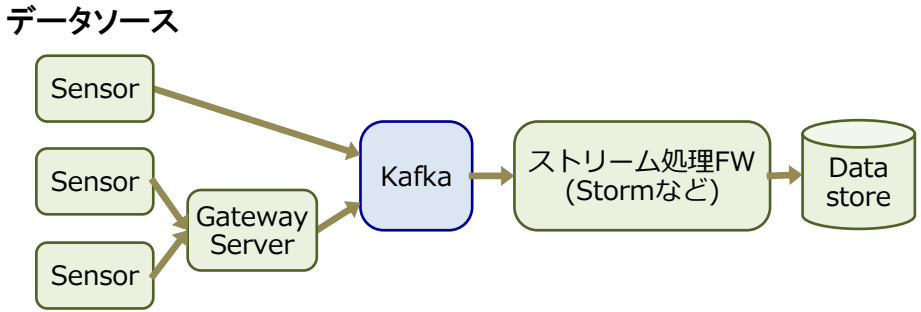
1. Apache Kafkaとは

Apache Kafkaとは

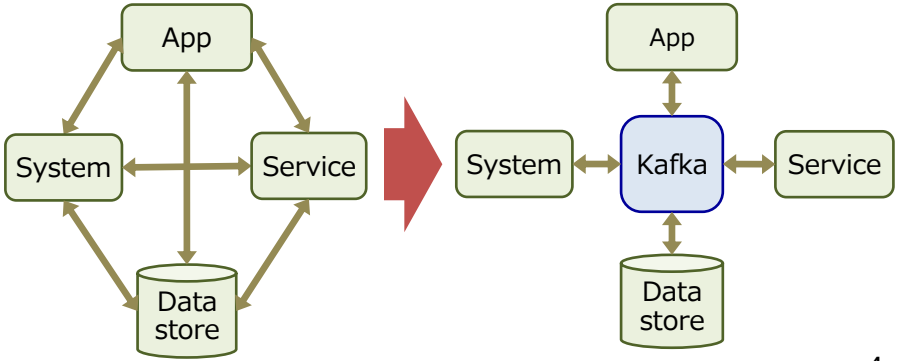
- スケーラビリティに優れた分散メッセージキュー
  - Pub/Subメッセージングモデルを採用
  - 書き込み/読み出し性能を重視しており、MQTTなどの標準プロトコルではなく、独自プロトコルを使用

- 主なユースケース：

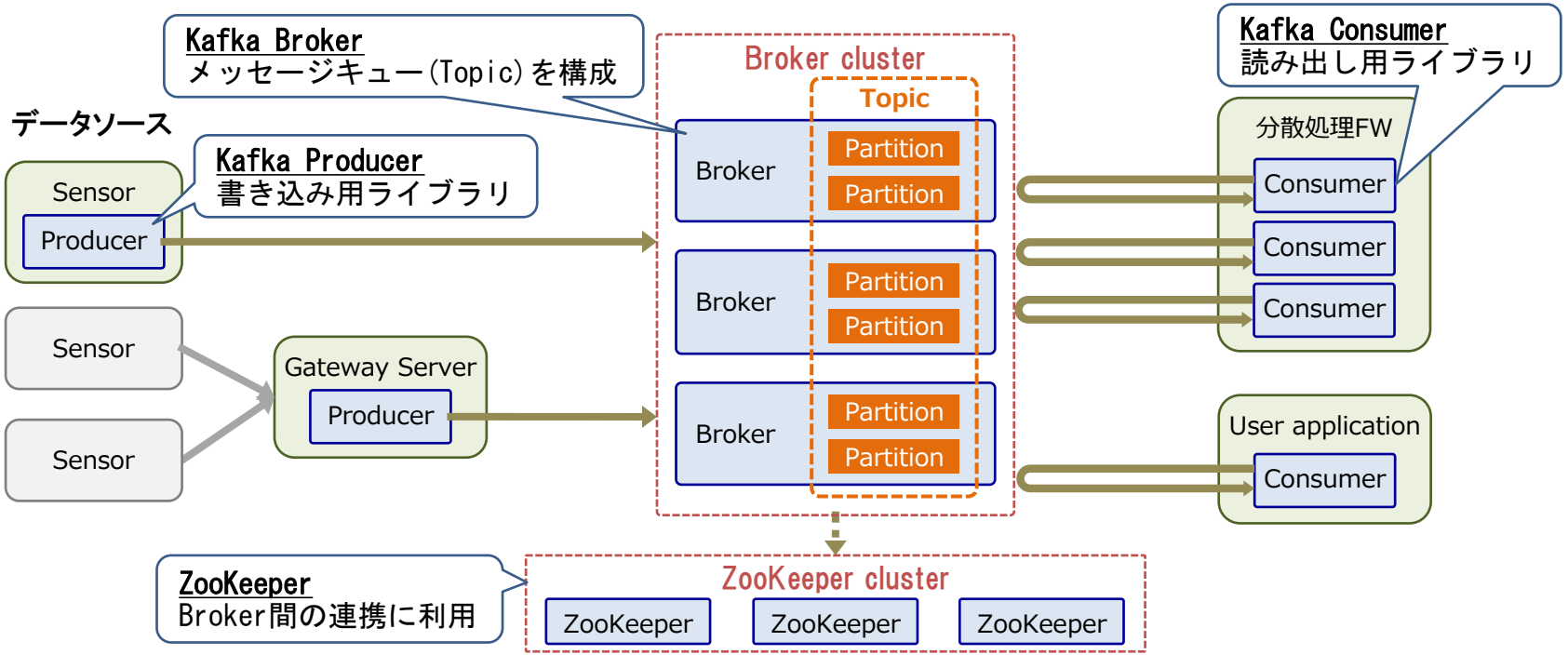
ストリームデータ処理システムにおけるキューイング



システム間で大量のデータを受け渡すためのパイプライン



Kafkaのシステム構成

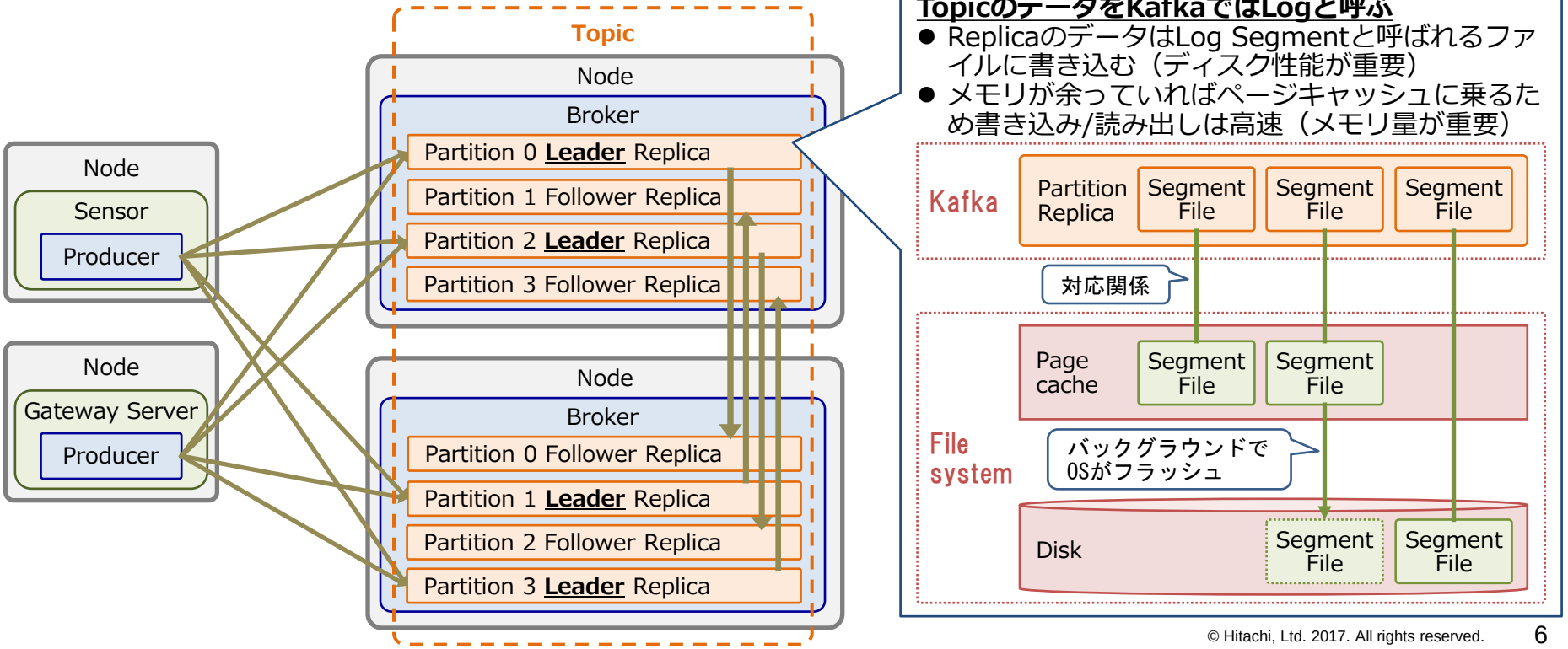


- Topic(1個の仮想的なキュー)を複数のBrokerに分散配置したPartitionで構成
- Topicのデータ書き込み/読み出しには Producer/Consumer ライブラリを使用
  - Java, C/C++, Pythonなど様々な言語用のライブラリが用意されている

データの書き込みと複製と保存

HITACHI
Inspire the Next

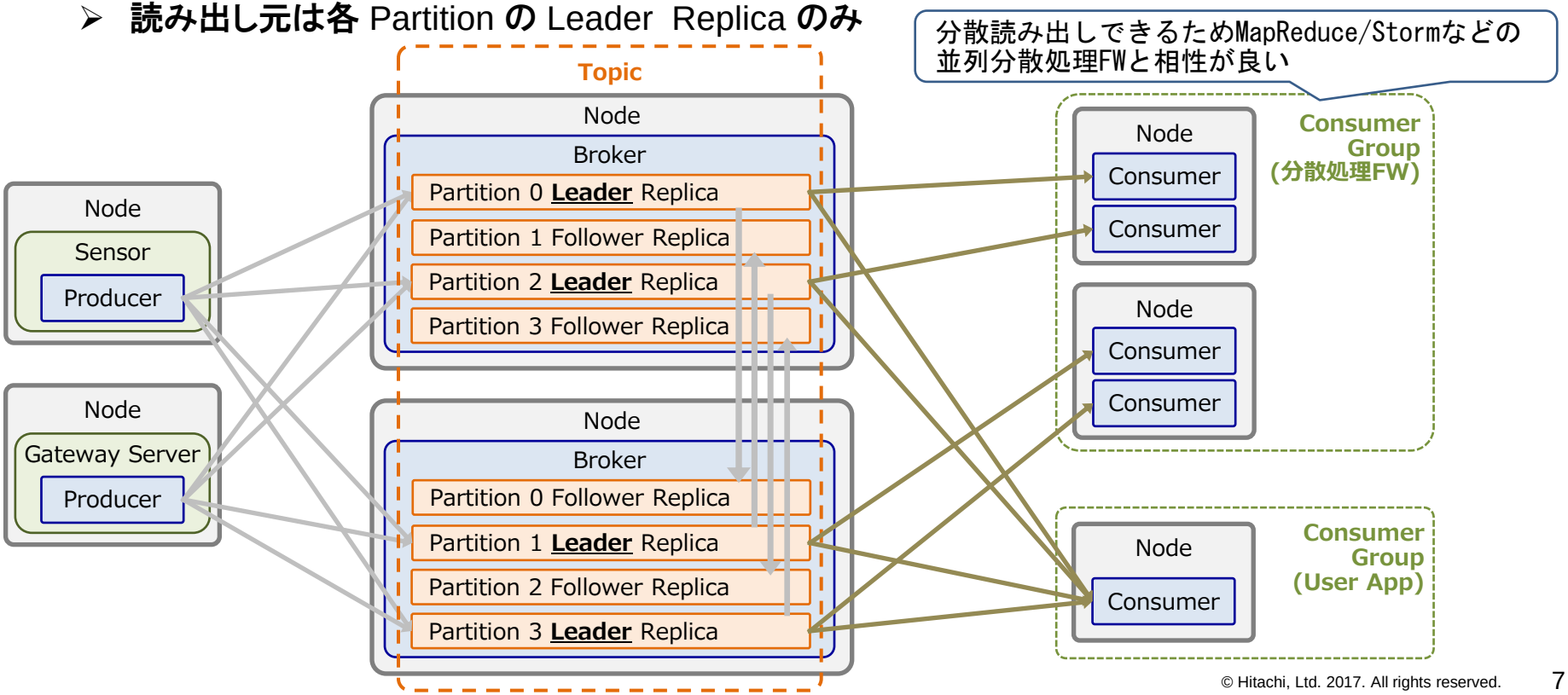
- **各Partitionは 1個のLeader Replica + 0個以上のFollower Replica で構成**
  - ➢ **Leader Replicaにデータを書き込み、別Broker上の Follower Replica に複製する**



データの読み出し

HITACHI
Inspire the Next

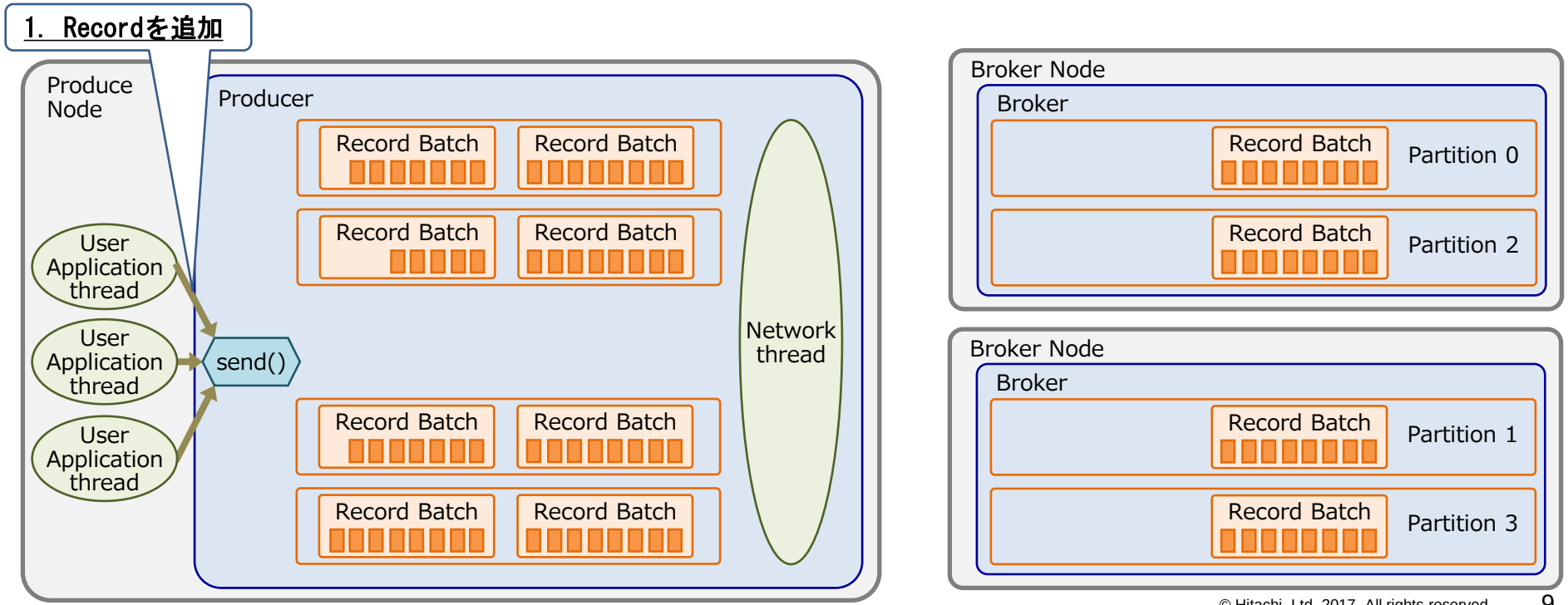
- ConsumerはPartition単位でデータを並列読み出し可能
  - 複数のConsumerでグループを構成し、1Topicのデータをグループ内のConsumerで分散読み出し
  - 読み出し元は各 Partition の Leader Replica のみ



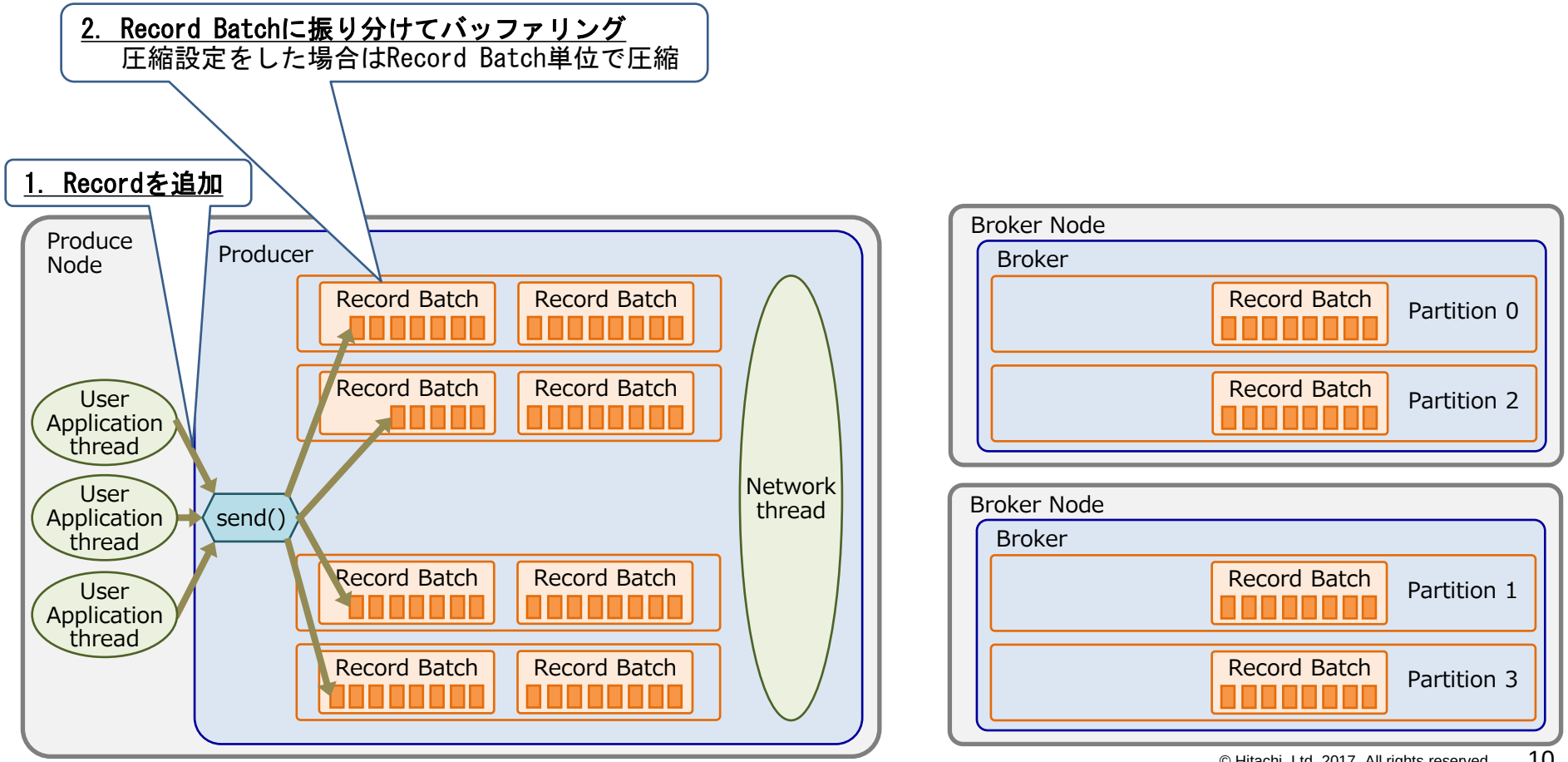
HITACHI
Inspire the Next

2. Kafkaの内部構造とチューニングポイント

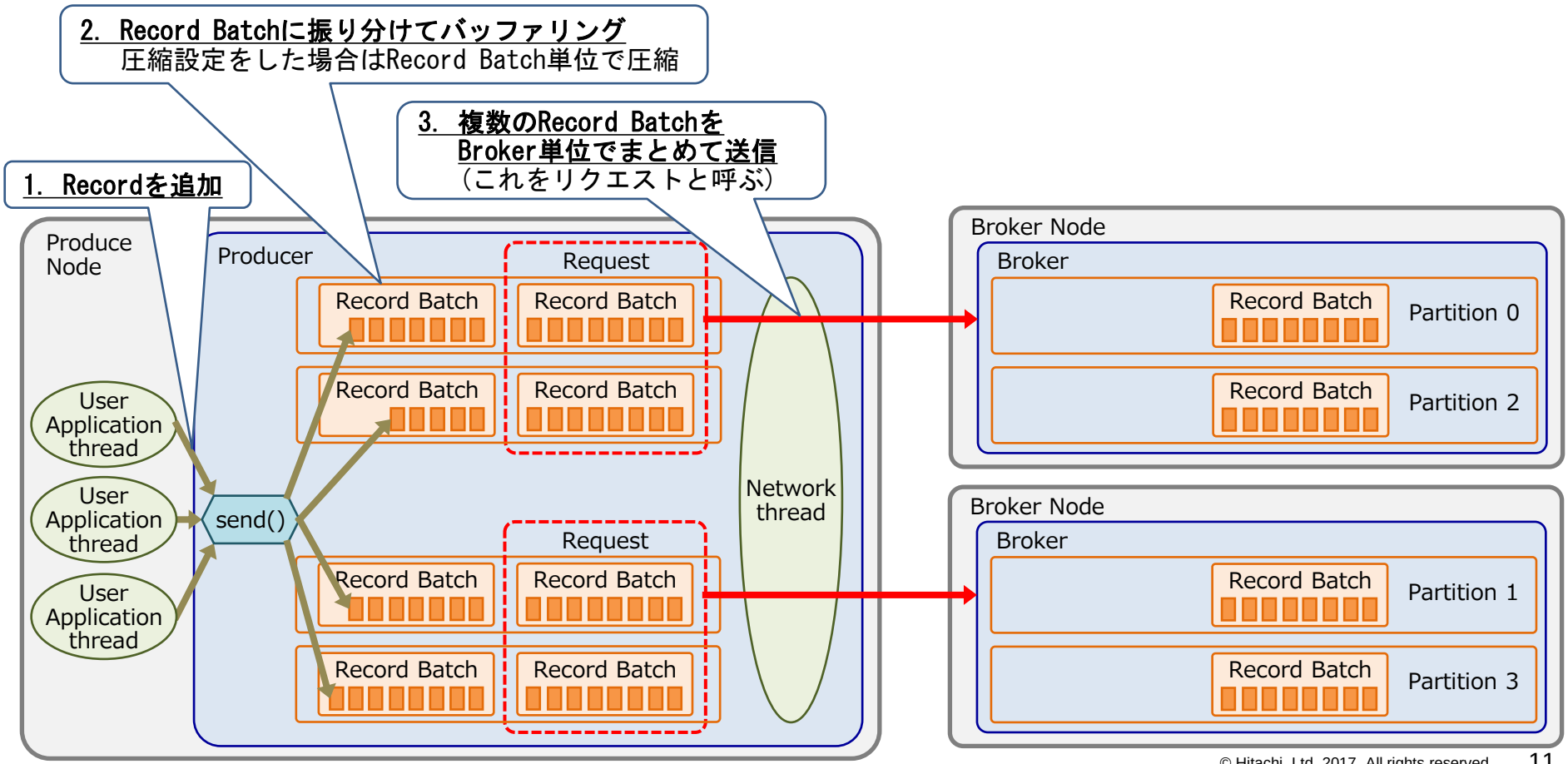
Producerの処理の流れ（1/5）



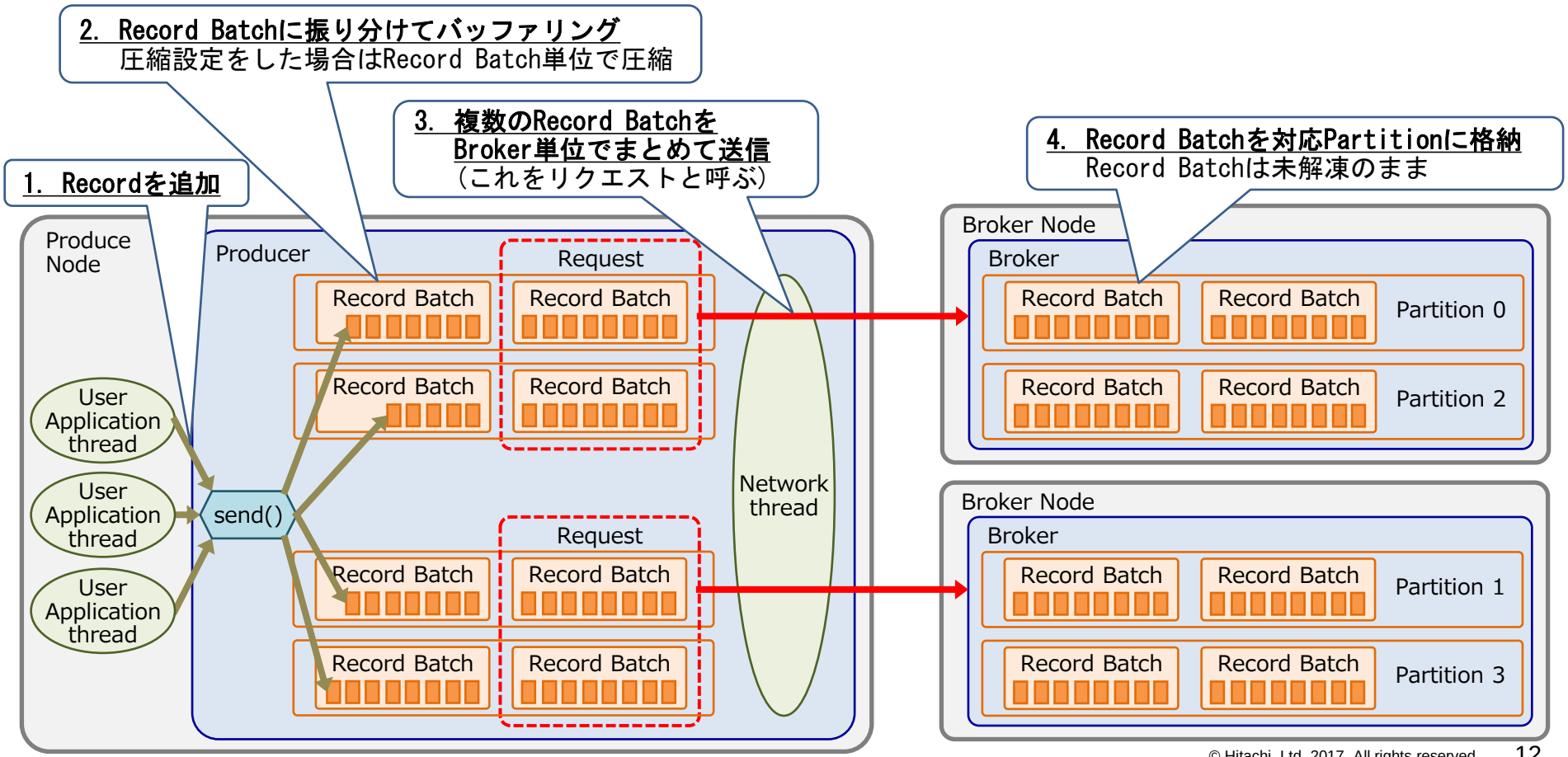
Producerの処理の流れ（2/5）



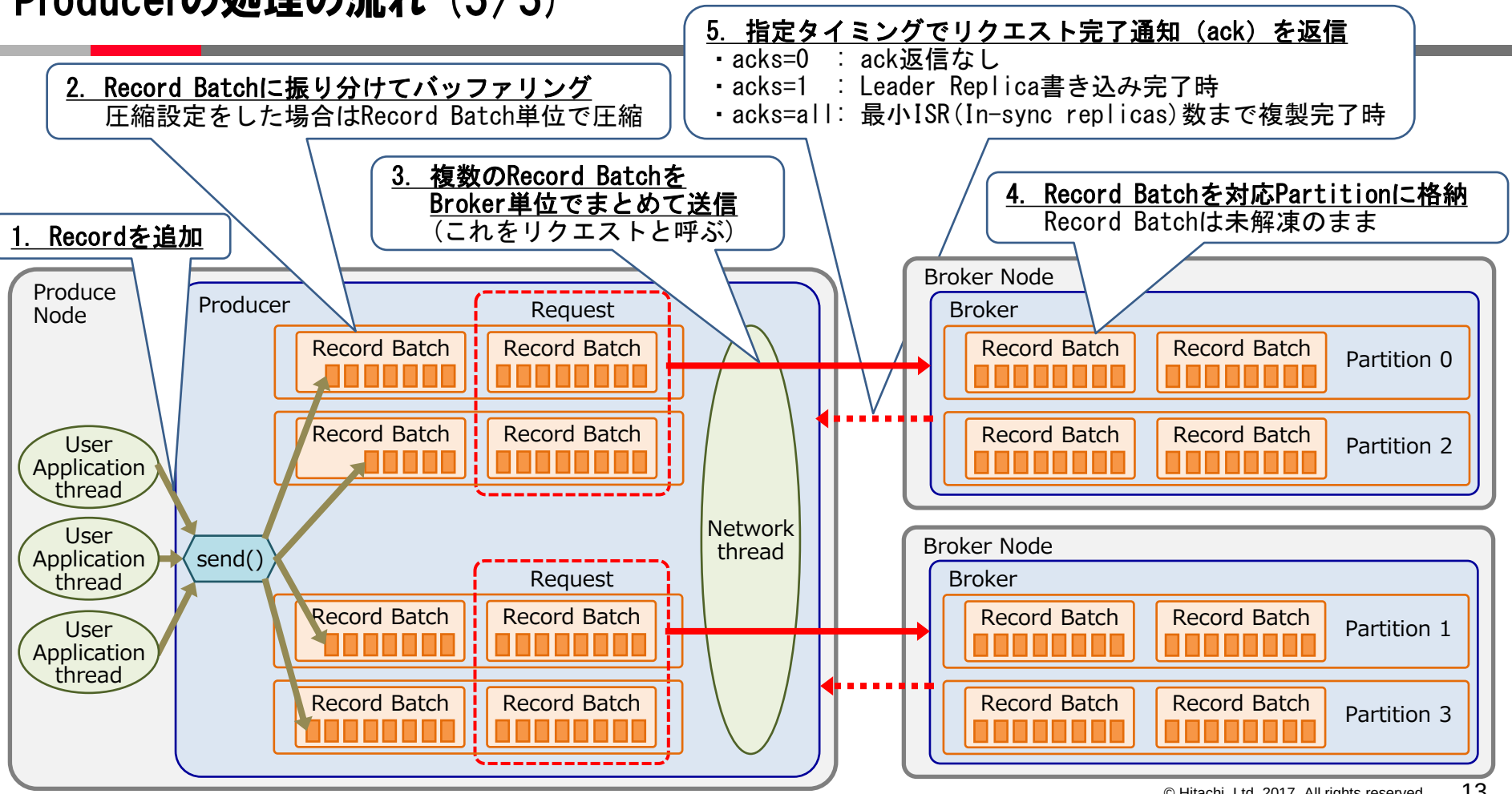
Producerの処理の流れ（3/5）



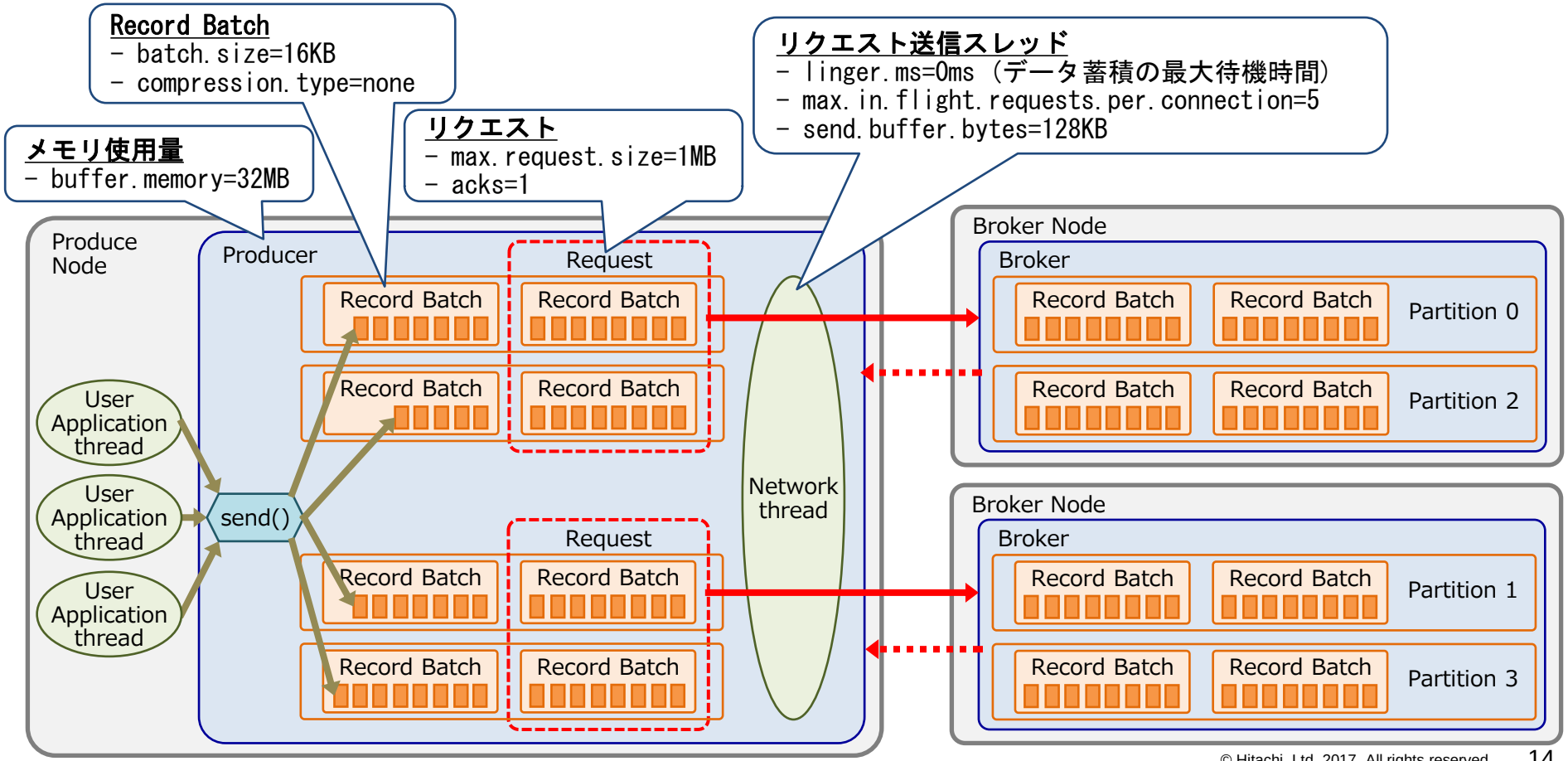
Producerの処理の流れ（4/5）



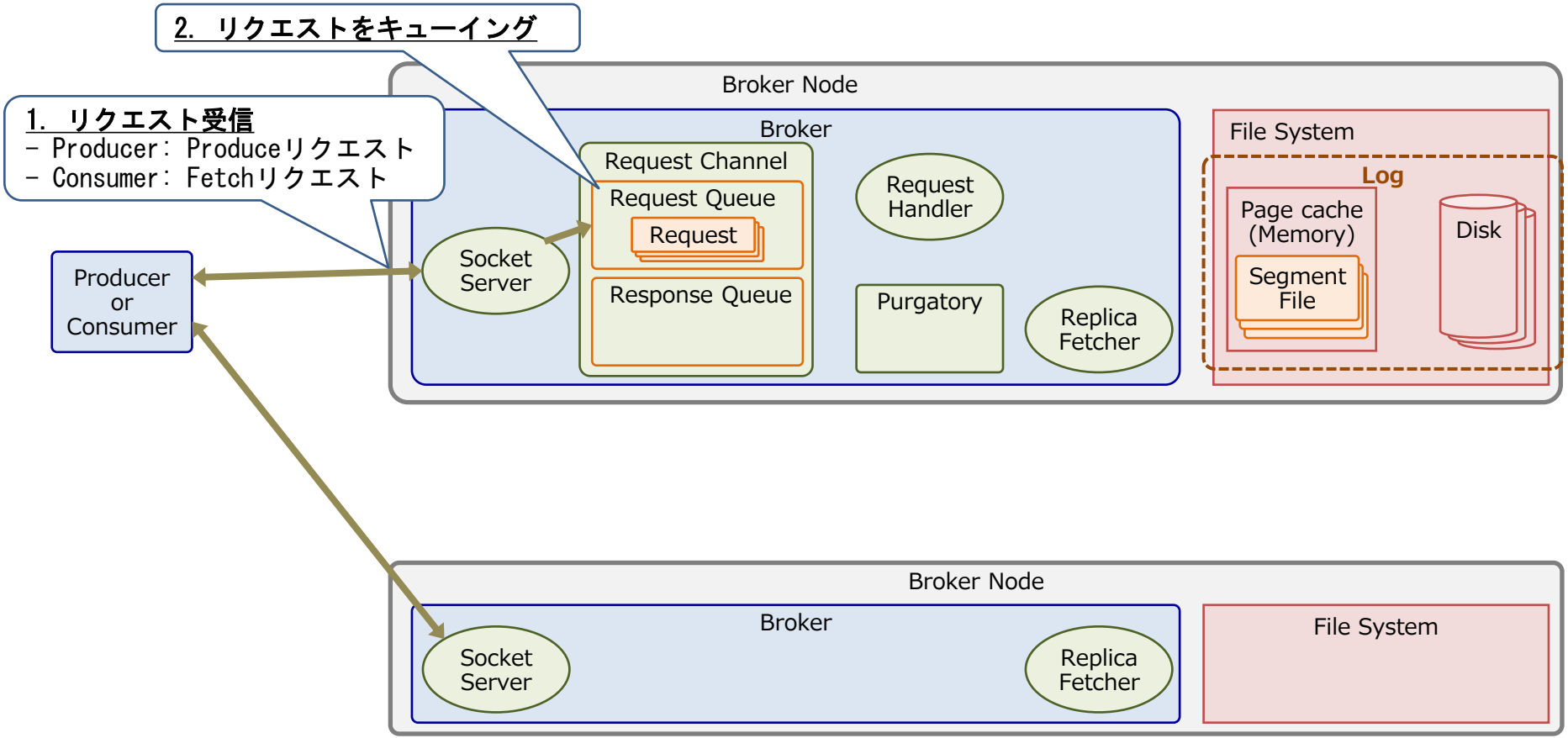
Producerの処理の流れ（5/5）



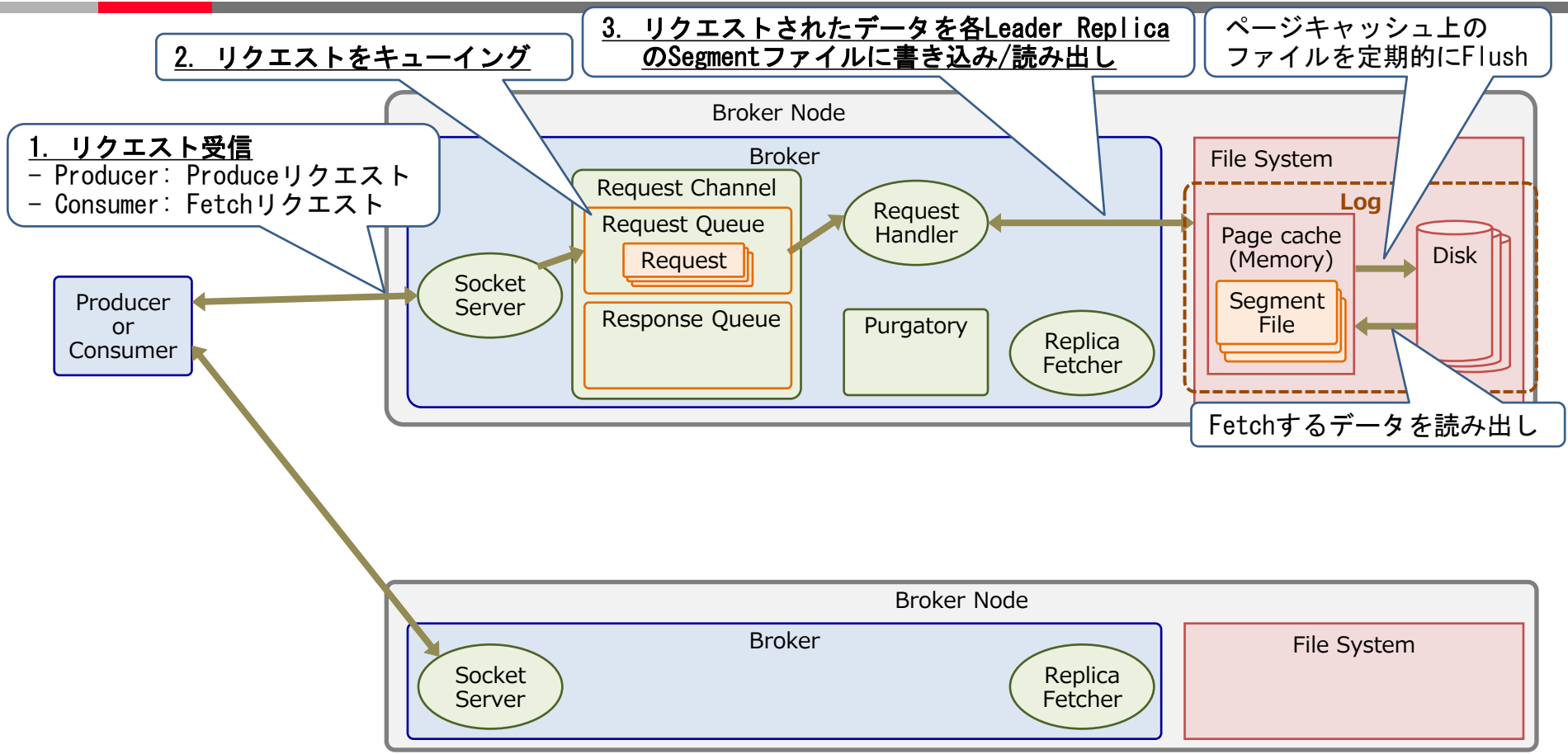
Producerのチューニングポイントとなるパラメータとデフォルト値



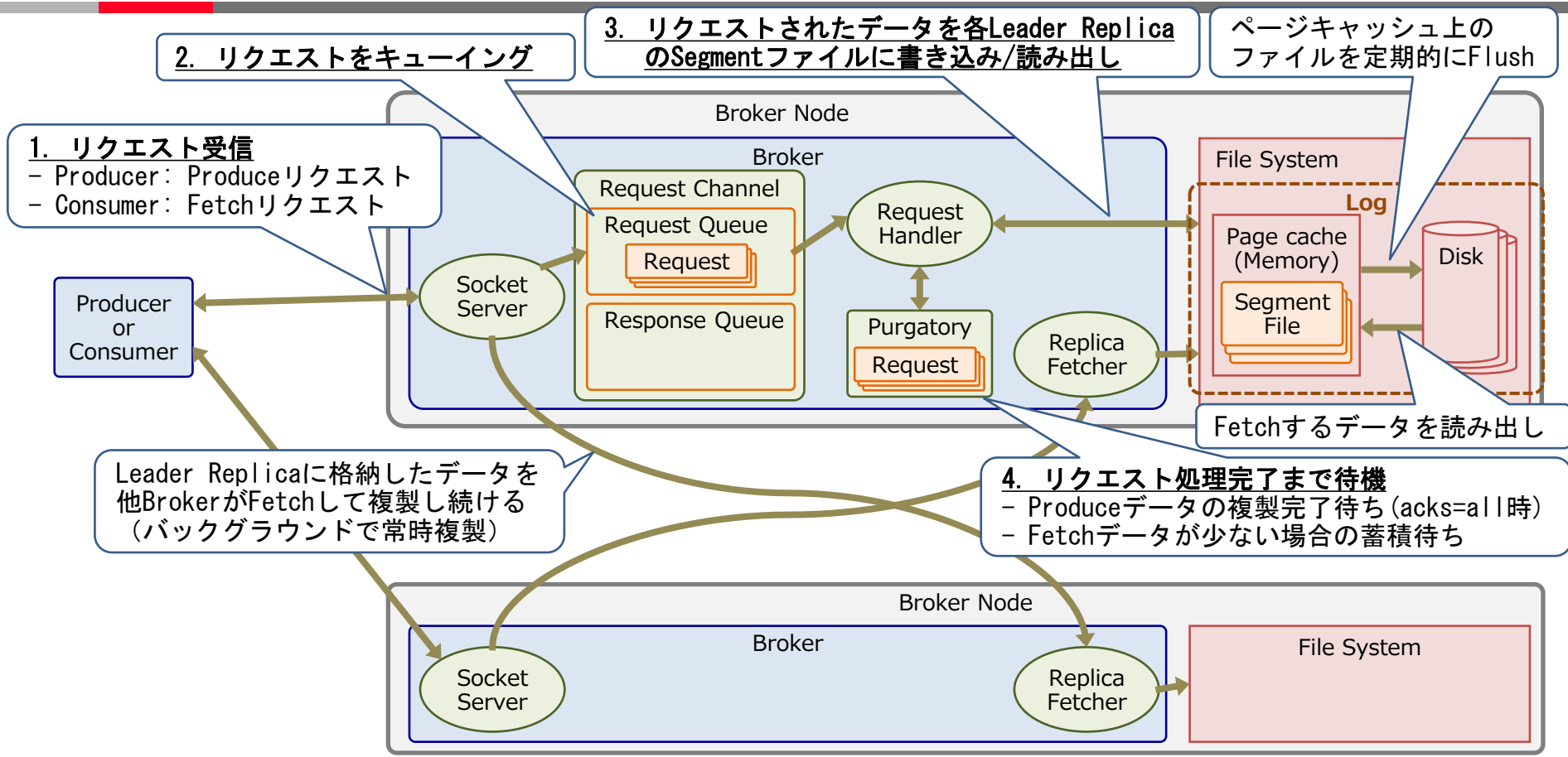
Brokerの処理の流れ［Produce（格納）/Fetch（取得）リクエスト時］（1/4）



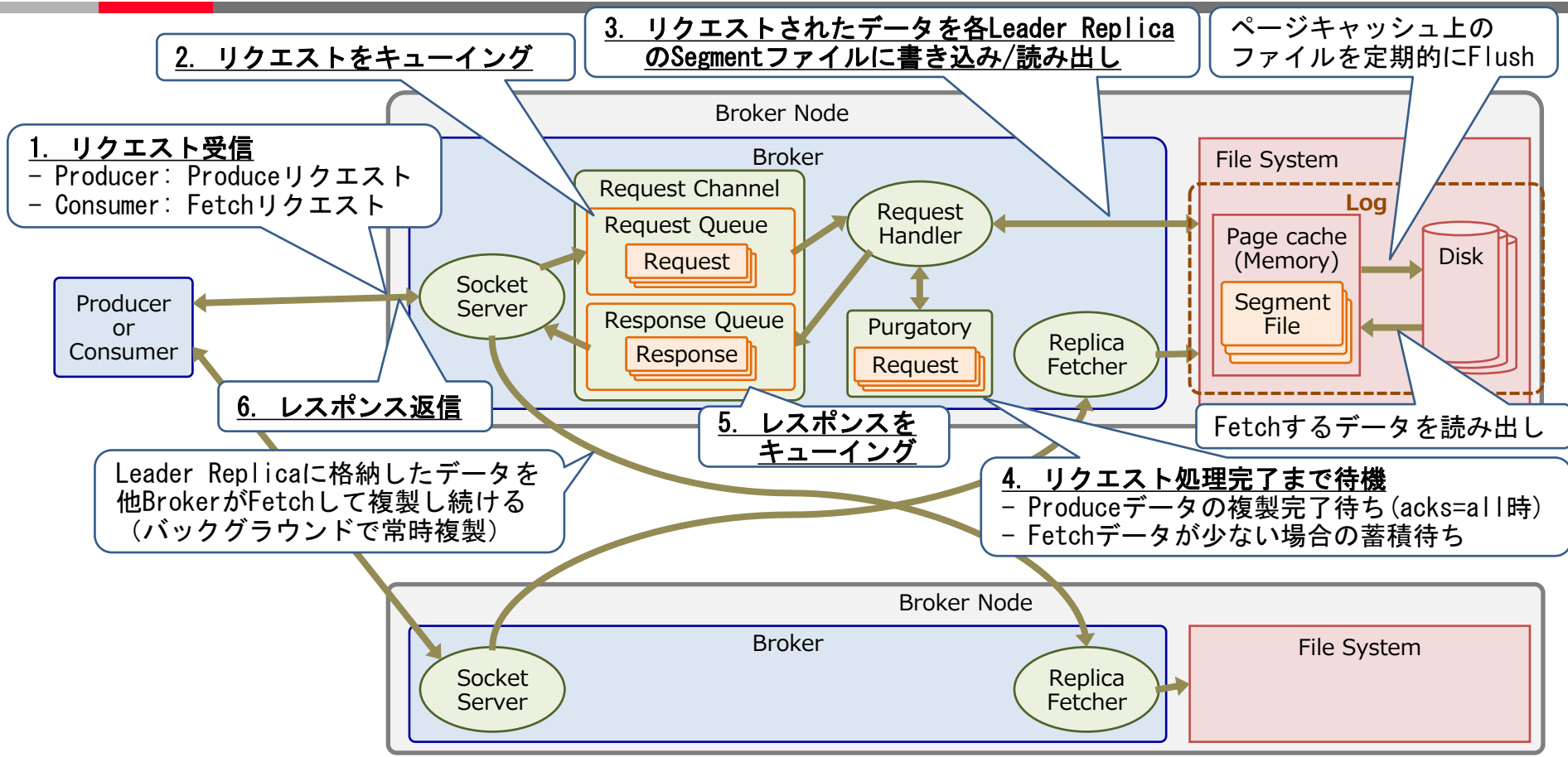
Brokerの処理の流れ［Produce（格納）/Fetch（取得）リクエスト時］（2/4）



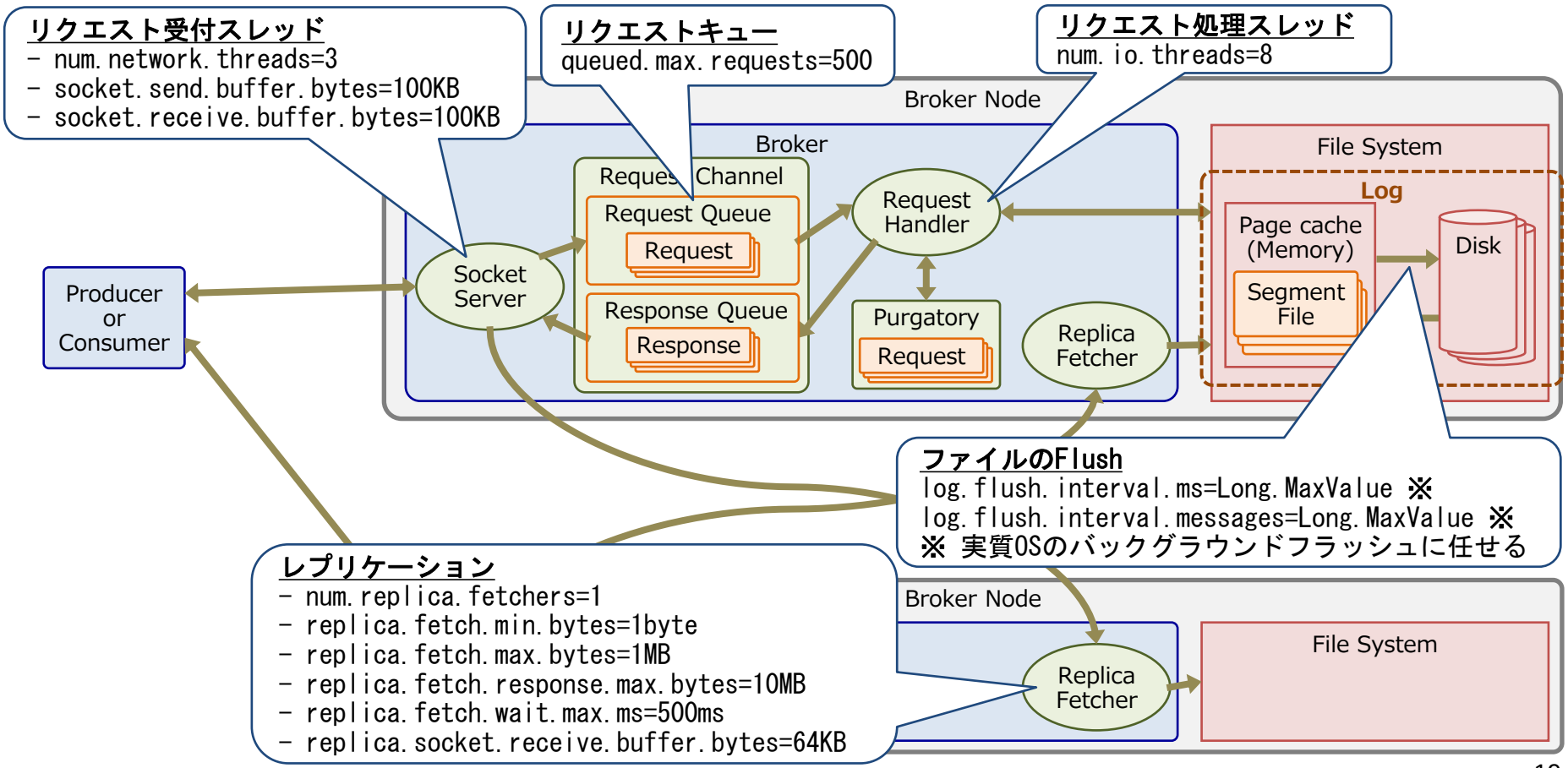
Brokerの処理の流れ［Produce（格納）/Fetch（取得）リクエスト時］（3/4）



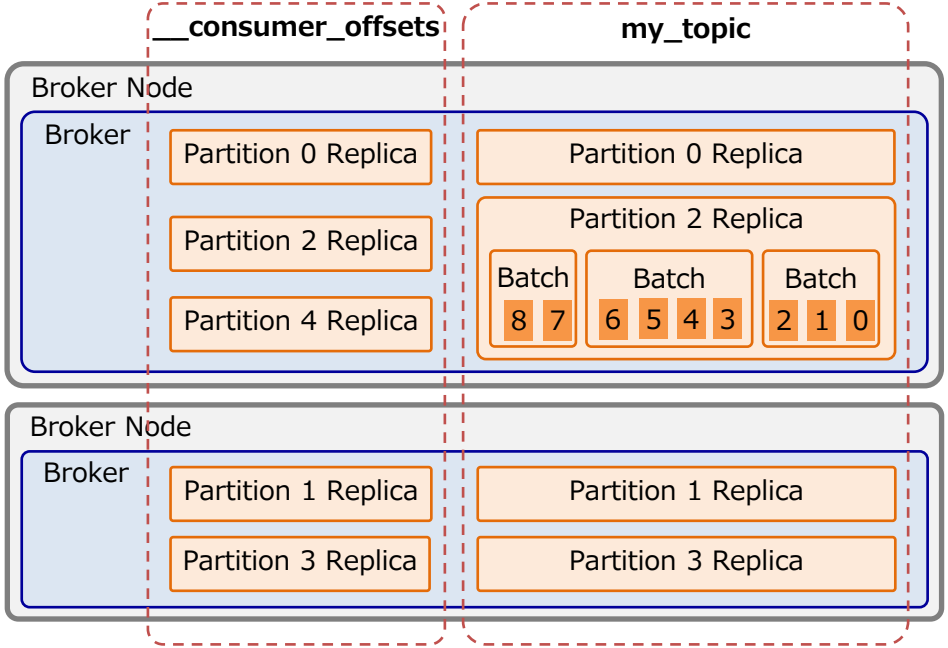
Brokerの処理の流れ［Produce（格納）/Fetch（取得）リクエスト時］（4/4）

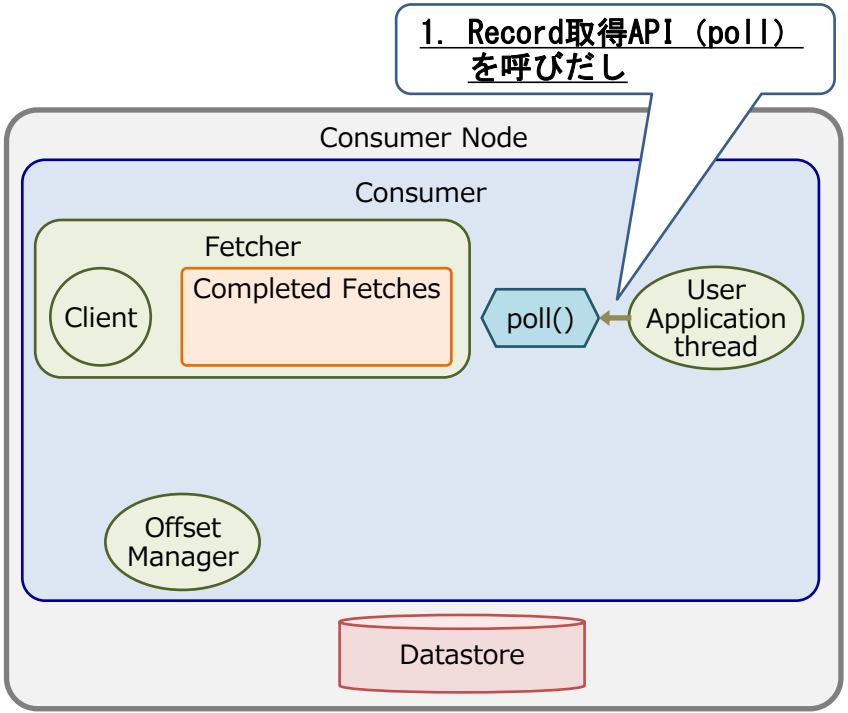


Brokerのチューニングポイントとなるパラメータとデフォルト値

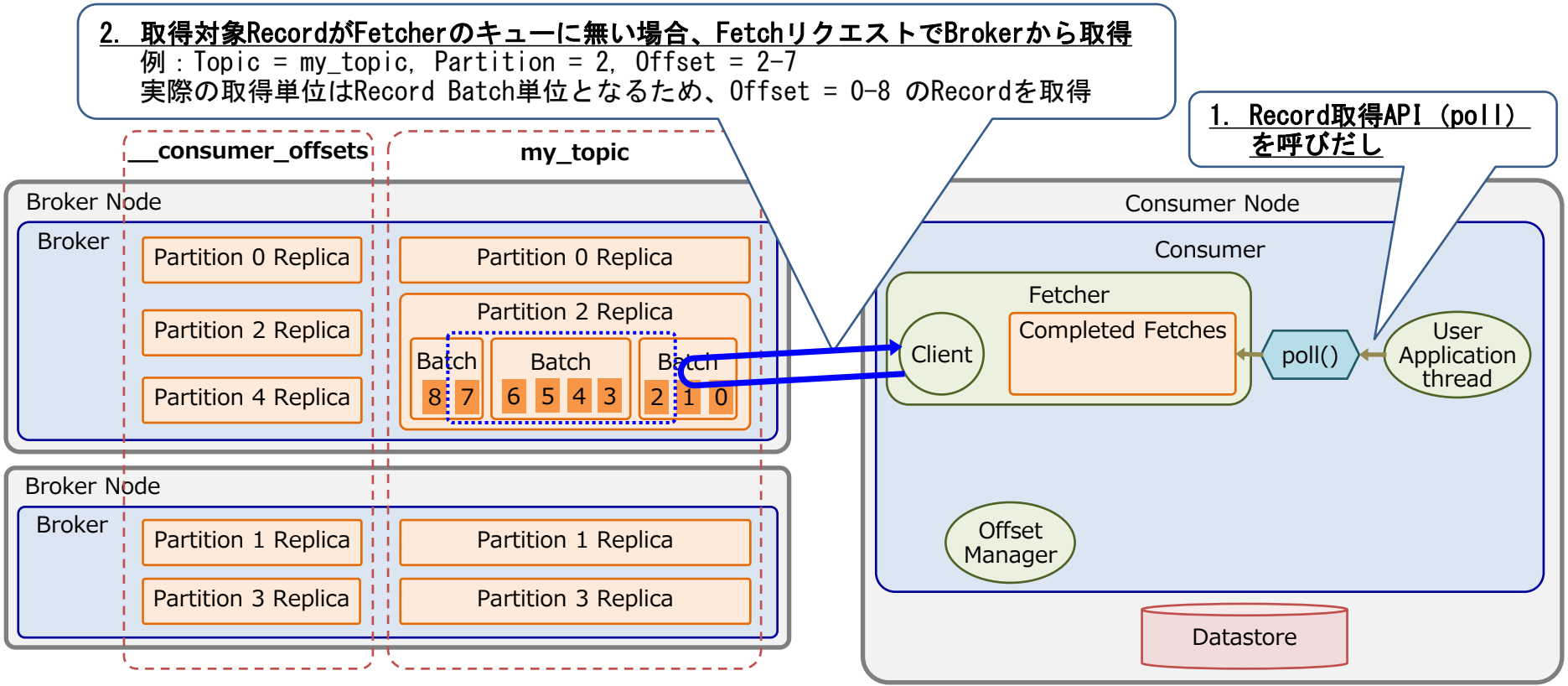


Consumerの処理の流れ(1/4)

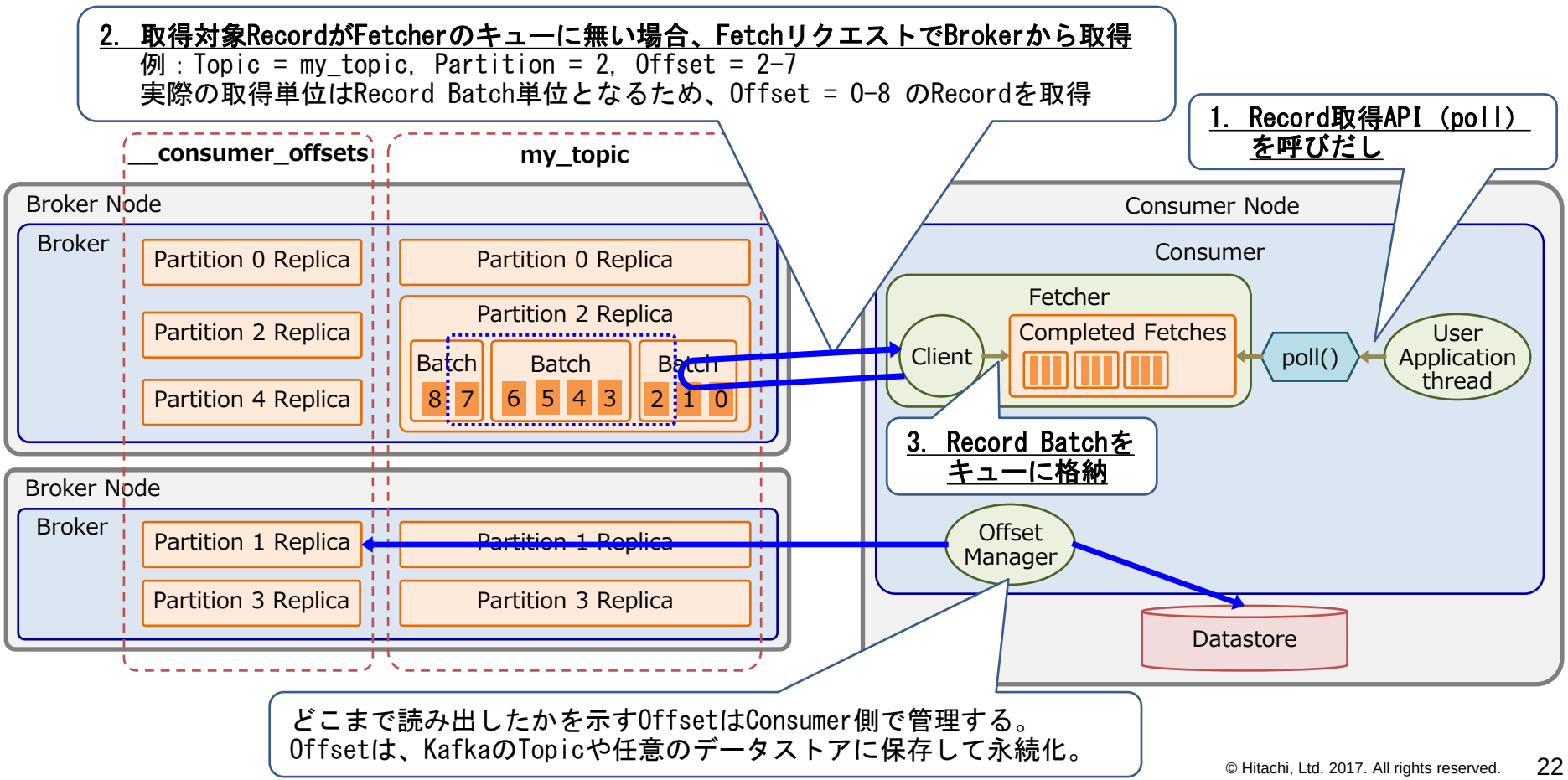




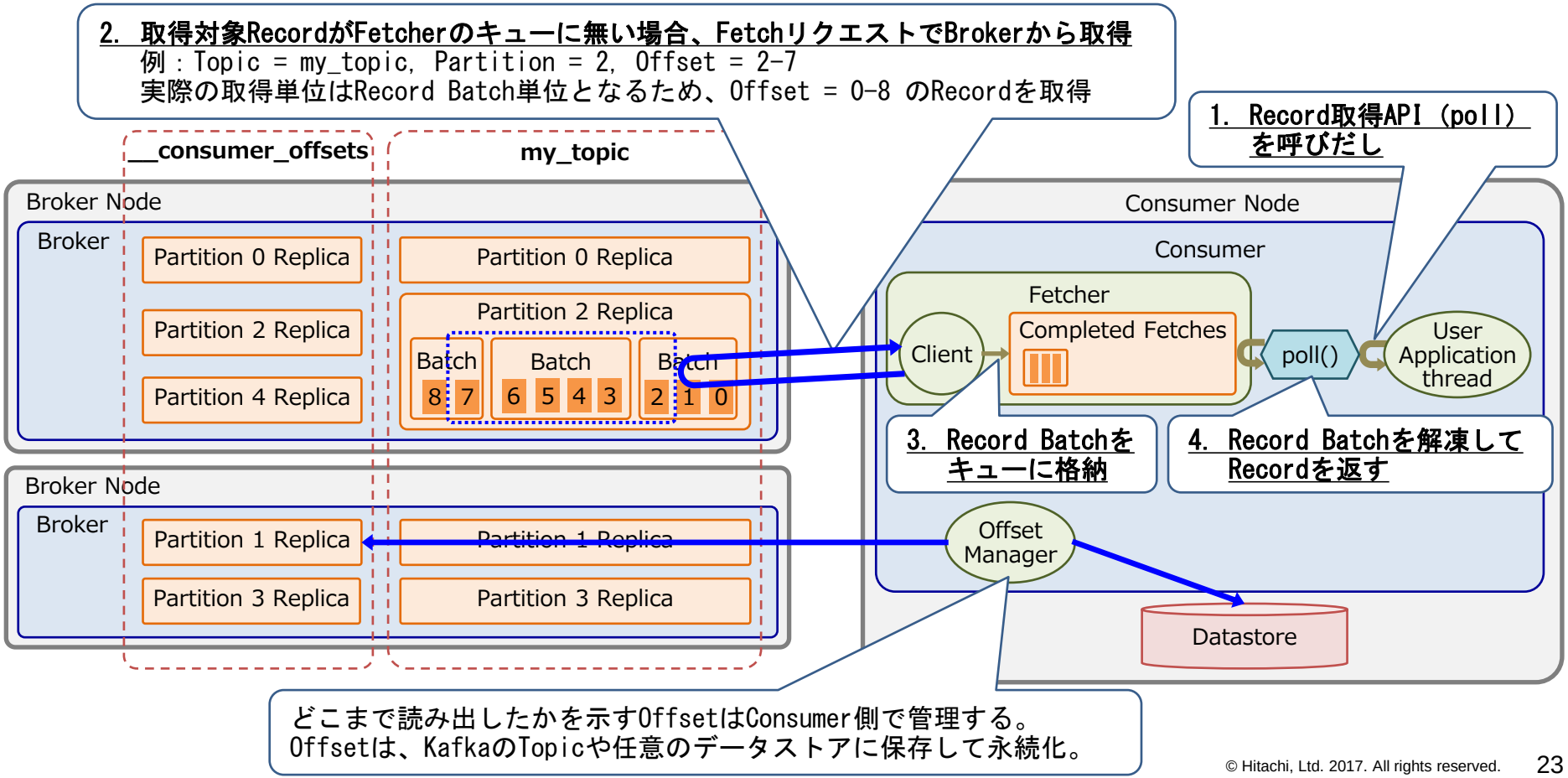
Consumerの処理の流れ(2/4)



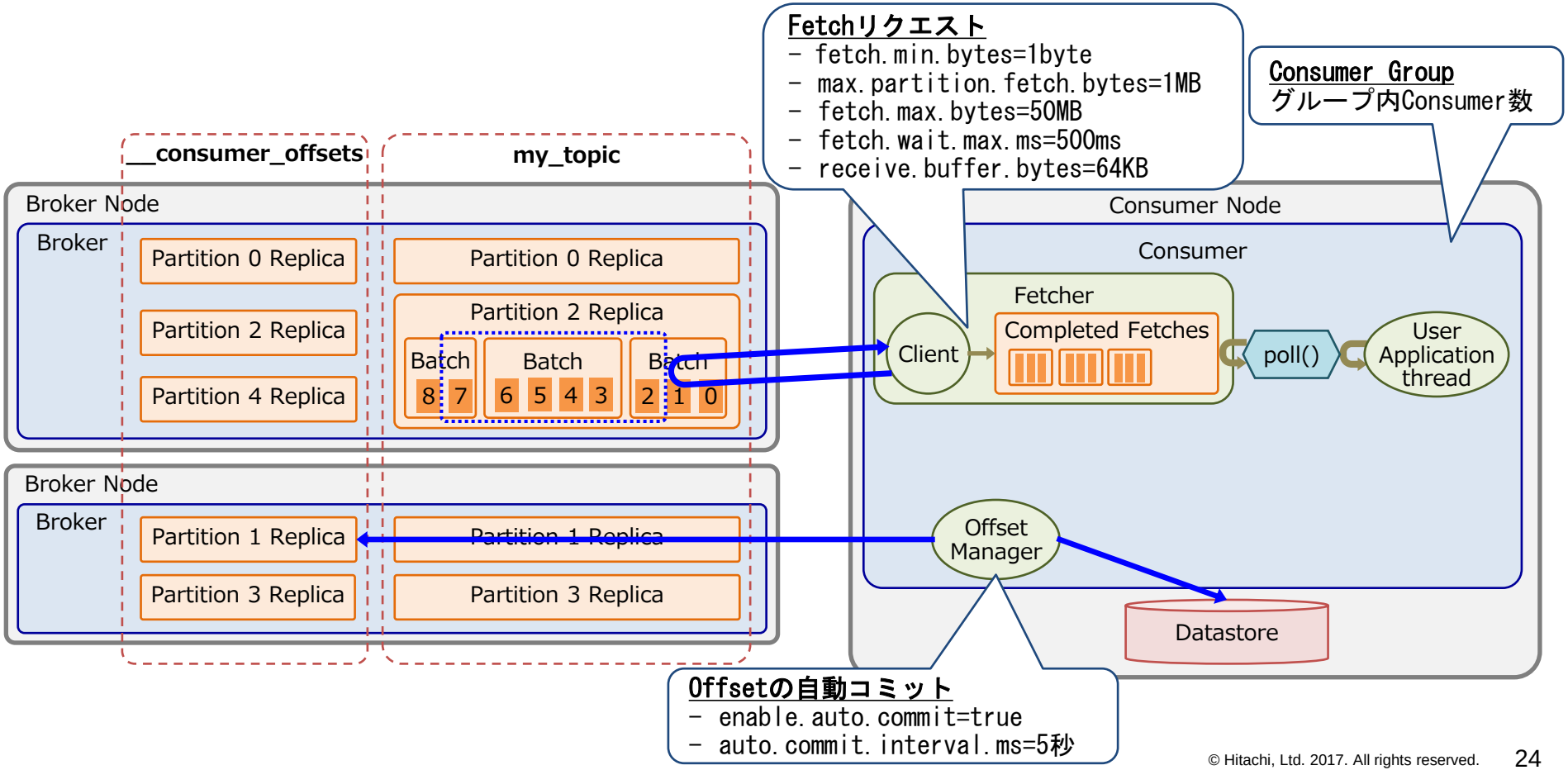
Consumerの処理の流れ(3/4)



Consumerの処理の流れ(4/4)



Consumerのチューニングポイントとなるパラメータとデフォルト値



HITACHI
Inspire the Next

3. 性能検証の概要

検証内容

HITACHI
Inspire the Next

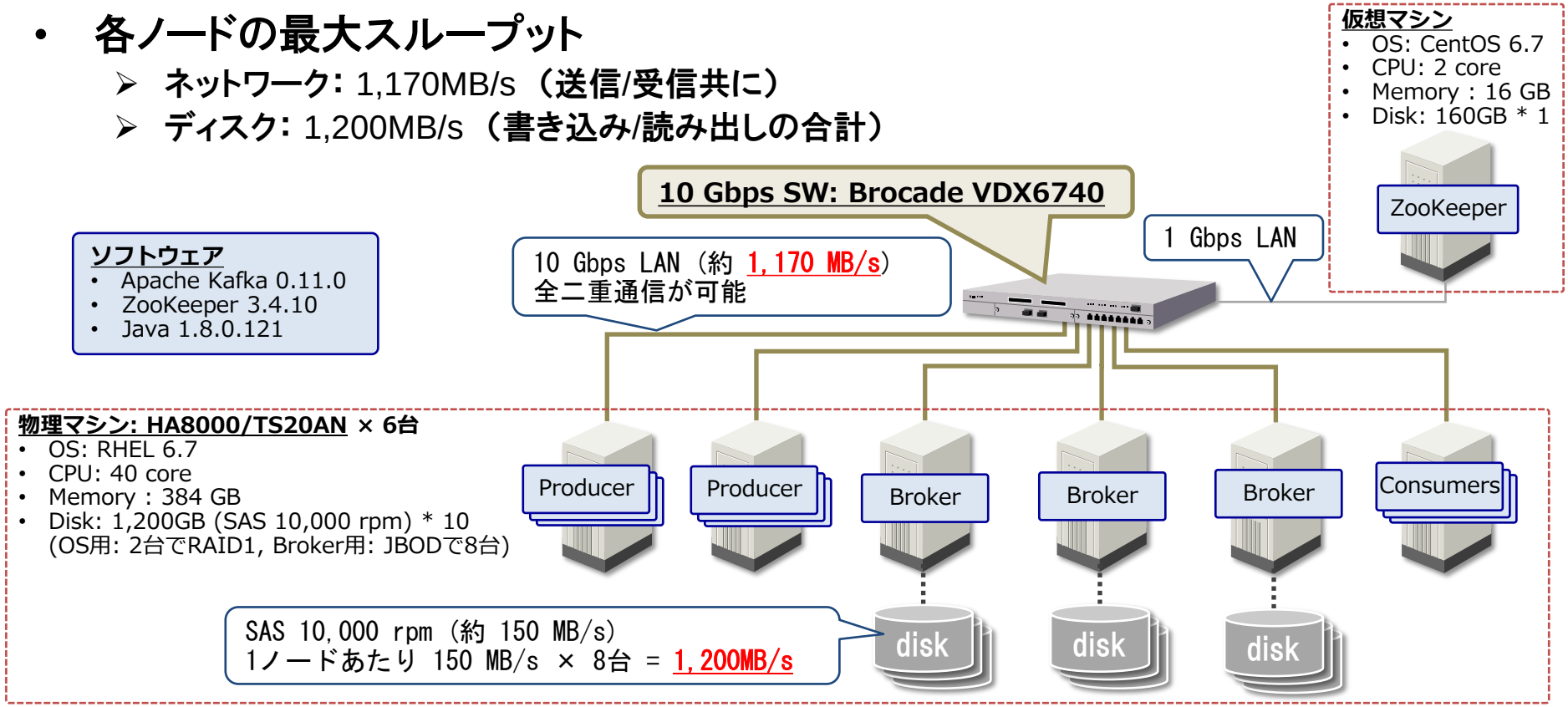
- **背景**
  - Kafkaは非同期レプリケーションであれば、ディスク/ネットワーク性能の上限に近いスループットを容易に出せる
  - しかし、同期レプリケーション時の性能については情報が少ない
    - 本番環境ではデータを保護するため同期レプリケーションが求められる

- **検証内容**
  - 同期レプリケーションでどこまでスループットを出せるか確認する
    - Producerノードで1KBのRecordを生成・送信し、1Consumer Groupが受信し続け、600秒間の平均スループットを測定
    - スループットが増加するとレイテンシも増加するが、チューニングでは考慮しない
  - 同期レプリケーションの設定
    - Replication factor = 3
    - 最小 In Sync Replica 数 = 2
    - acks = all
      - Producerはデータが最小ISR数(2個)以上のReplicaに同期されたことを確認する(Leader Replica + 1個以上のFollower Replicaに複製されたことを確認)

システム構成とディスク/ネットワーク性能

HITACHI
Inspire the Next

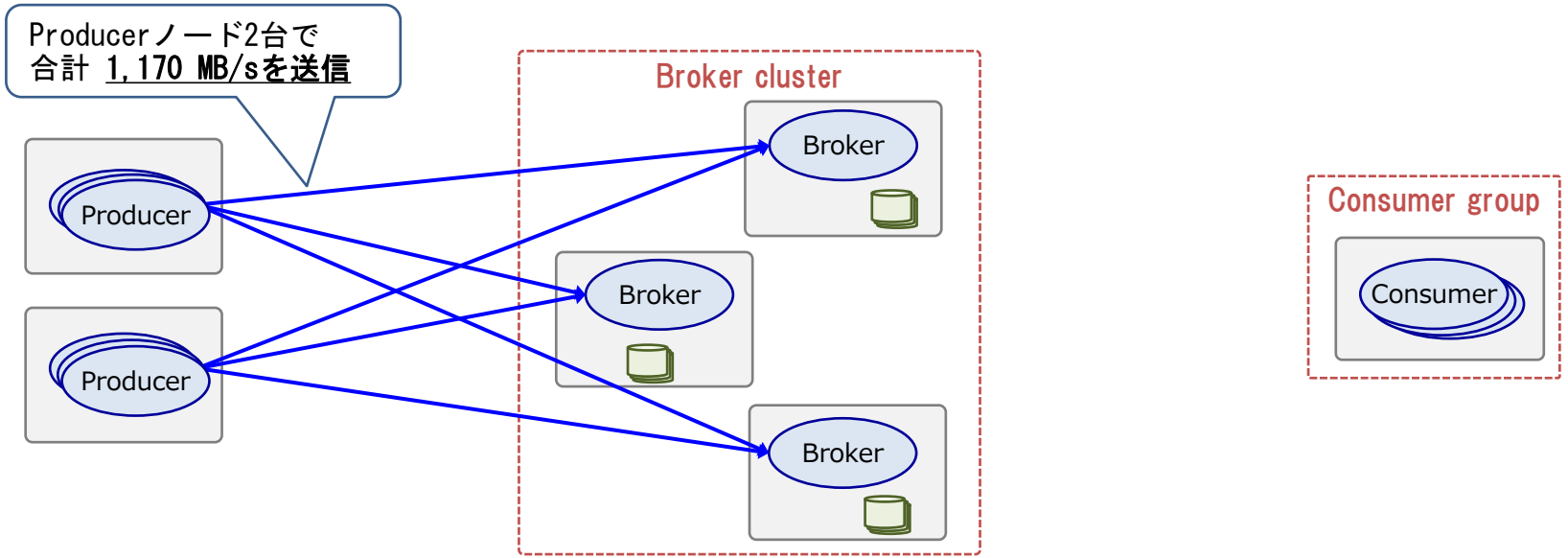
- 各ノードの最大スループット
  - ネットワーク: 1,170MB/s（送信/受信共に）
  - ディスク: 1,200MB/s（書き込み/読み出しの合計）



理論上の最大スループット（1/3）

HITACHI
Inspire the Next

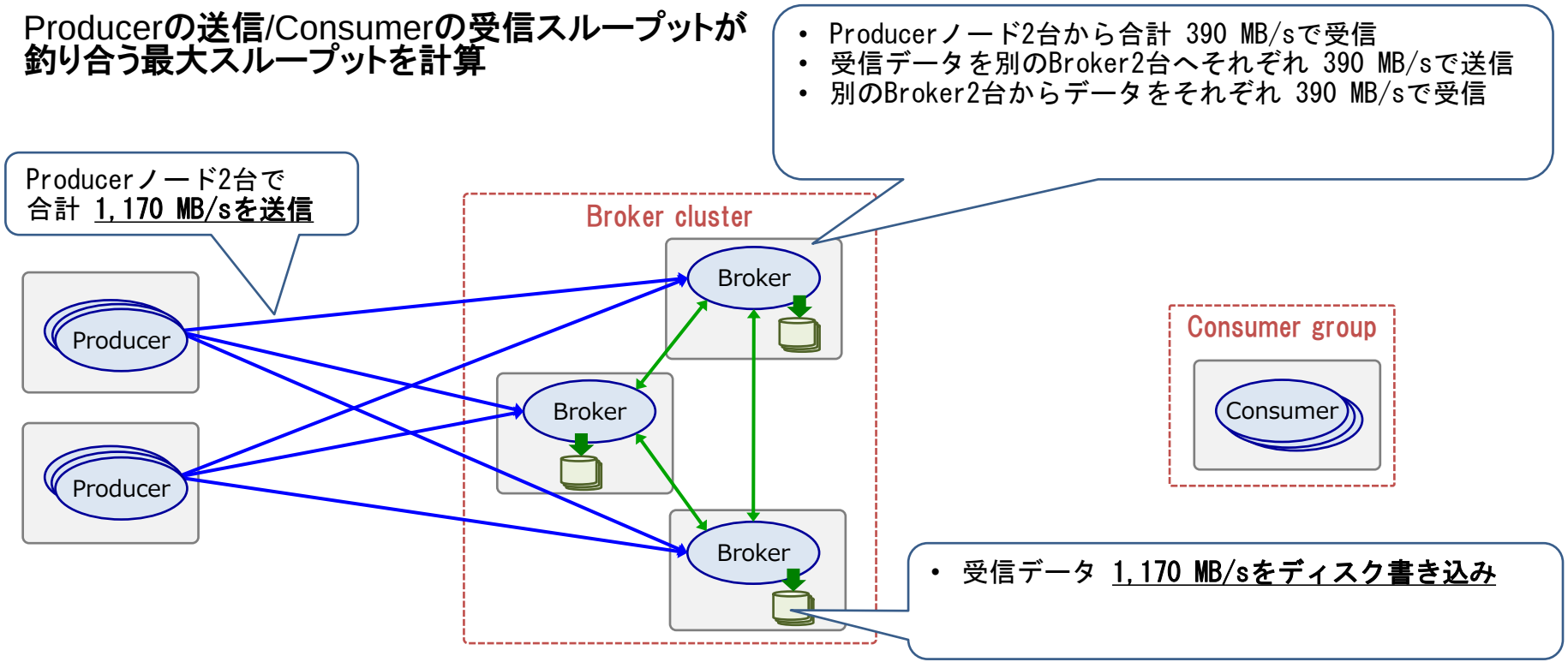
- Producerの送信/Consumerの受信スループットが釣り合う最大スループットを計算



理論上の最大スループット（2/3）

HITACHI
Inspire the Next

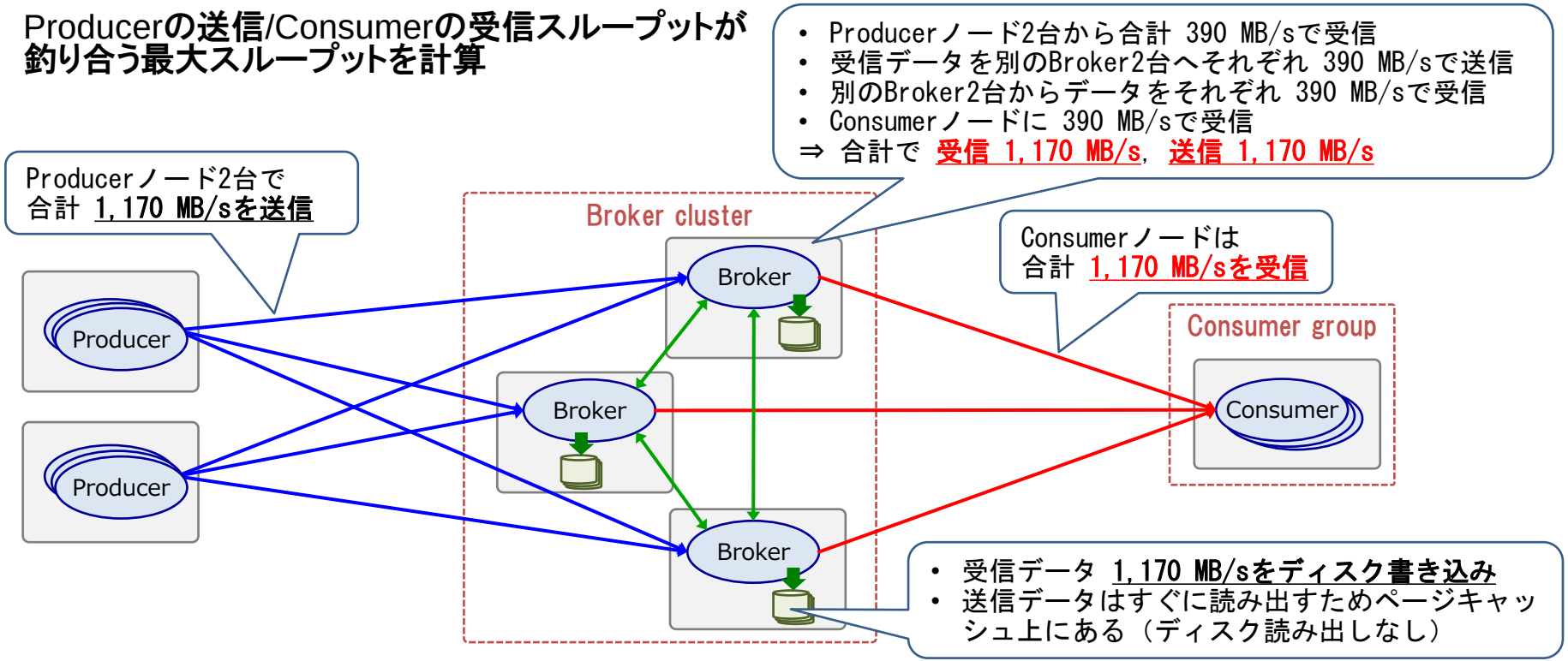
- Producerの送信/Consumerの受信スループットが釣り合う最大スループットを計算



理論上の最大スループット（3/3）

HITACHI
Inspire the Next

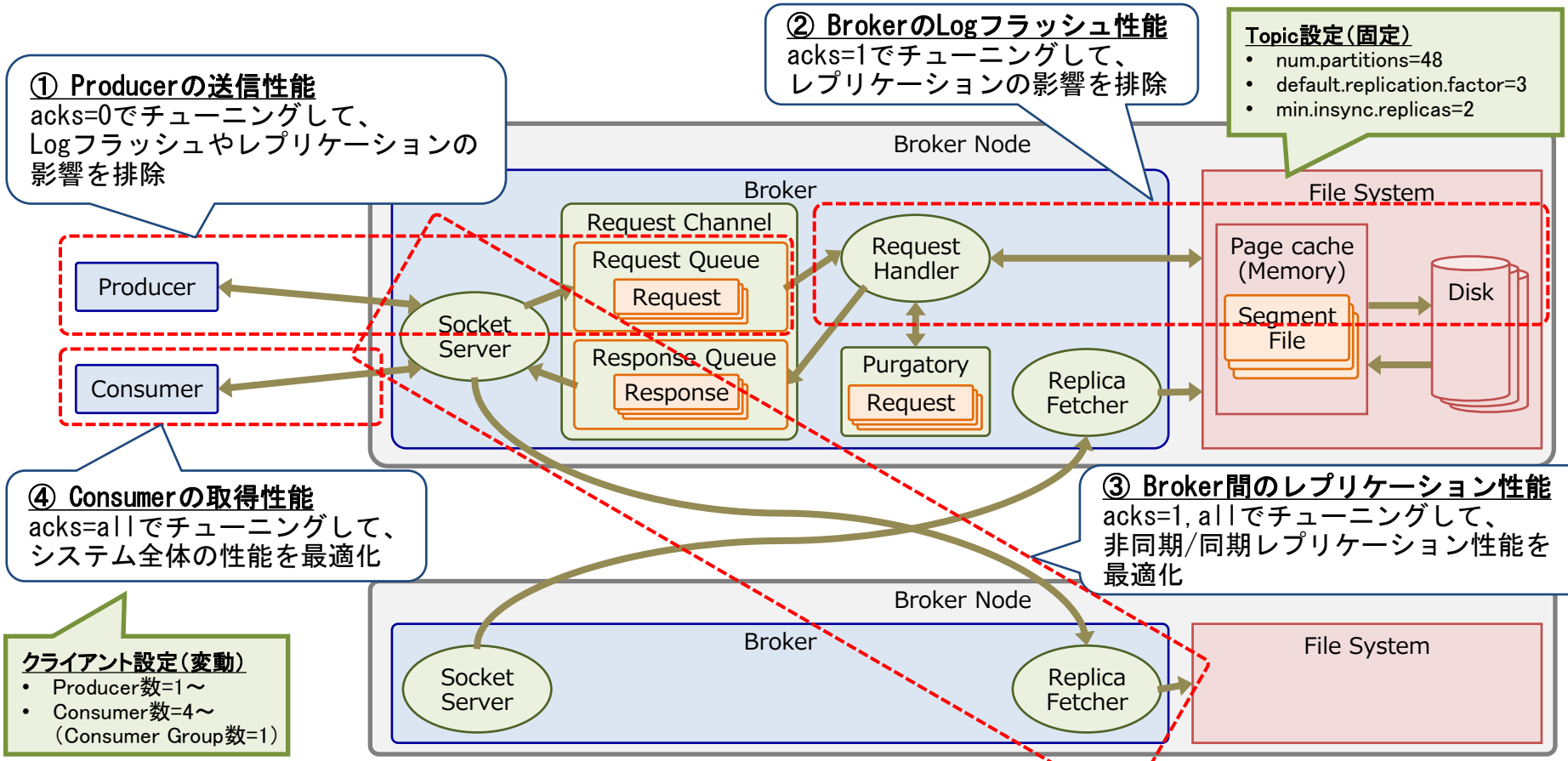
- Producerの送信/Consumerの受信スループットが釣り合う最大スループットを計算



Brokerの送受信とConsumerの受信がボトルネック： システム全体の最大スループットは 1,170 MB/s

検証範囲を4分割して順番にチューニング

HITACHI
Inspire the Next



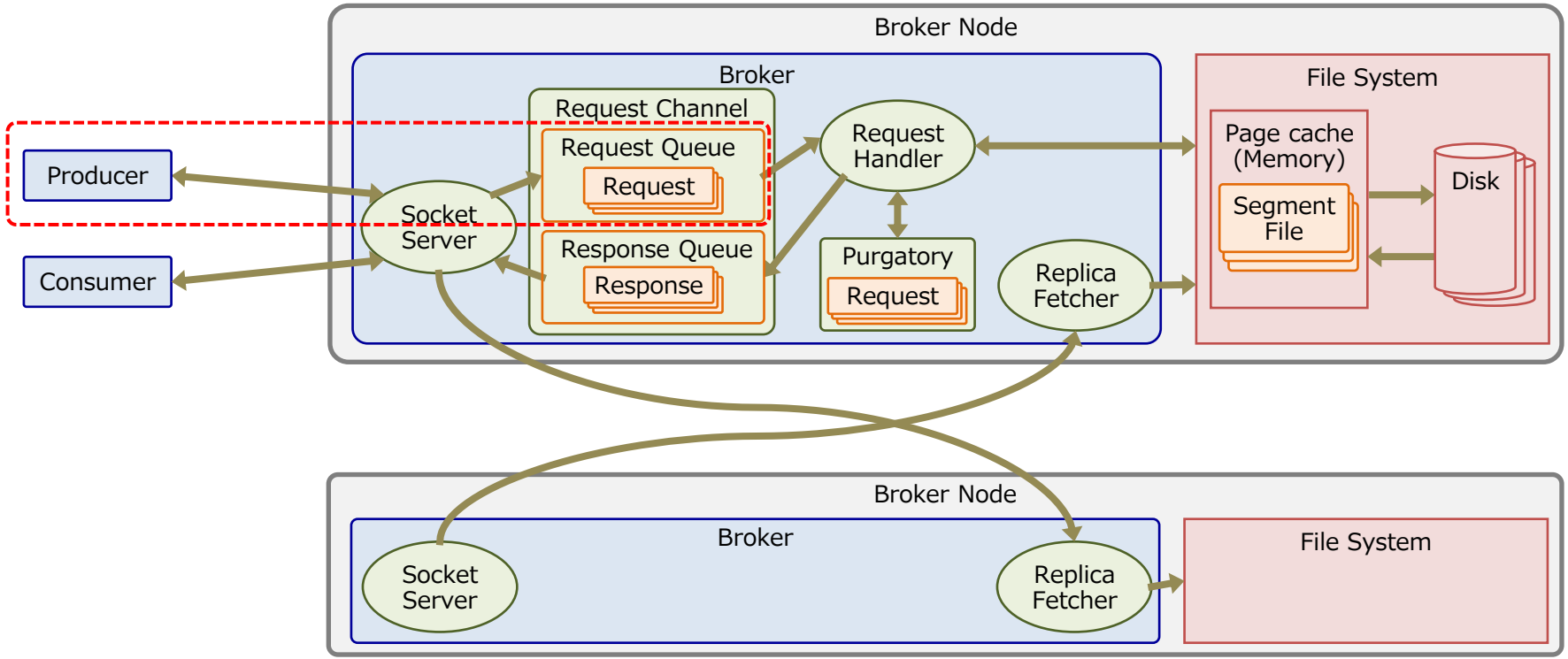
HITACHI
Inspire the Next

4. 検証結果と考察

① Producerの送信性能のチューニング

HITACHI
Inspire the Next

- **目的：** acks=0 で Produceスループットを最大化する
  - ➢ Brokerに十分な量のデータを入力するため、Producer送信スループット1,170MB/sをめざす



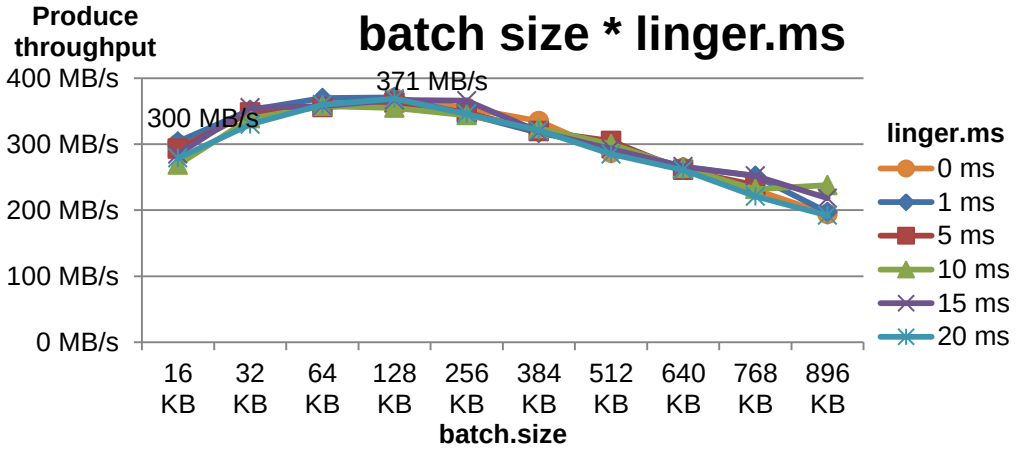
① Producerの送信性能のチューニング

HITACHI
Inspire the Next

- 送信するRecord Batchサイズは、最大サイズ(batch.size)と蓄積待機時間(linger.ms)で決まるため、これを同時にチューニング
  - Broker側のLogフラッシュやレプリケーションの影響を排除するため acks=0 で測定
  - Producer用ノード1台かつProducer1個で測定

Producerのパラメータ設定
- acks = 0
- buffer.memory = 32MB->1GB
- max.request.size = 1MB->最大値
- max.in.flight.requests.per.connection = 5->最大値
- max.block.ms = 60秒->最大値
- retries = 0回->最大値
- request.timeout.ms = 30秒->最大値

※ 最大値： IntegerまたはLong型の最大値を設定



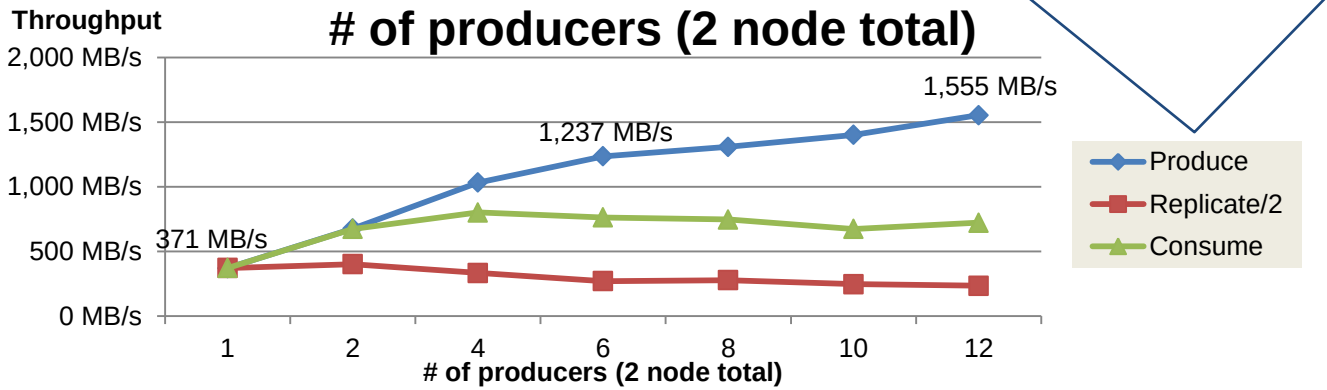
- batch.sizeのチューニングは効果が大きいが、linger.msの影響は少なかった
- batch.size=128KB, linger.ms=1msのとき、Produceスループットは 300 -> 371MB/s に向上

① Producerの送信性能のチューニング

HITACHI
Inspire the Next

- Producer用ノード2台でProducer数を2の倍数で増やして測定

- **Produce**: Producerの送信スループット
- **Replicate/2**: Broker間レプリケーションのスループット
  （同じデータを他の2台に複製するため、1/2の値を表示）
- **Consume**: Consumerの受信スループット

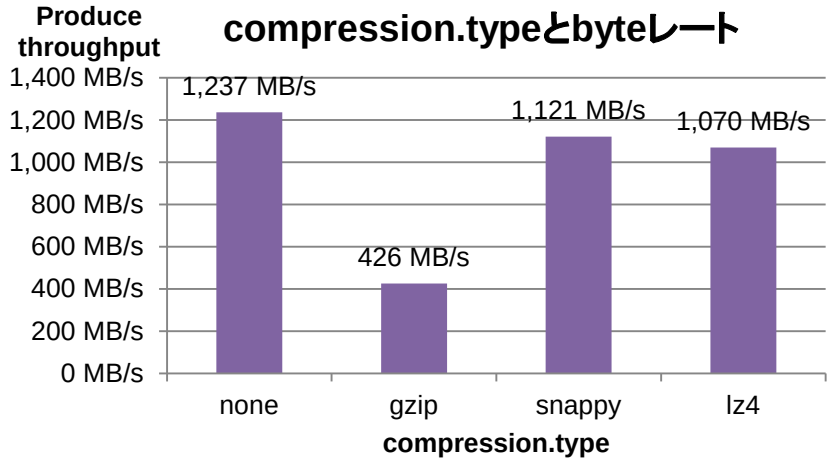


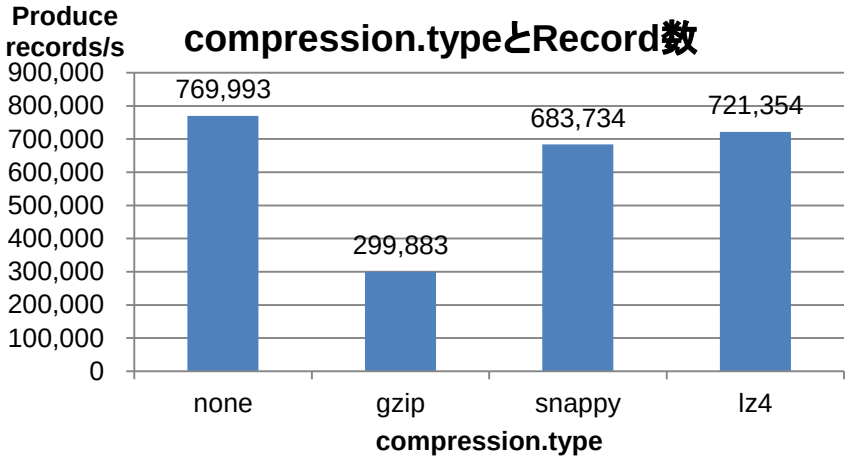
Producer数=6 でProduceスループットが理論値である 1,170 MB/sを超えた
これは、Replicateスループットが遅延した分のネットワーク帯域を使用できるためと考えられる

① Producerの送信性能のチューニング

HITACHI
Inspire the Next

- ProducerでRecord Batchを圧縮して測定
  - 4種類の圧縮アルゴリズム(compression.type)を検証
  - 圧縮率により送信レコード数が変化するため、ByteレートとRecordレートの両方を測定



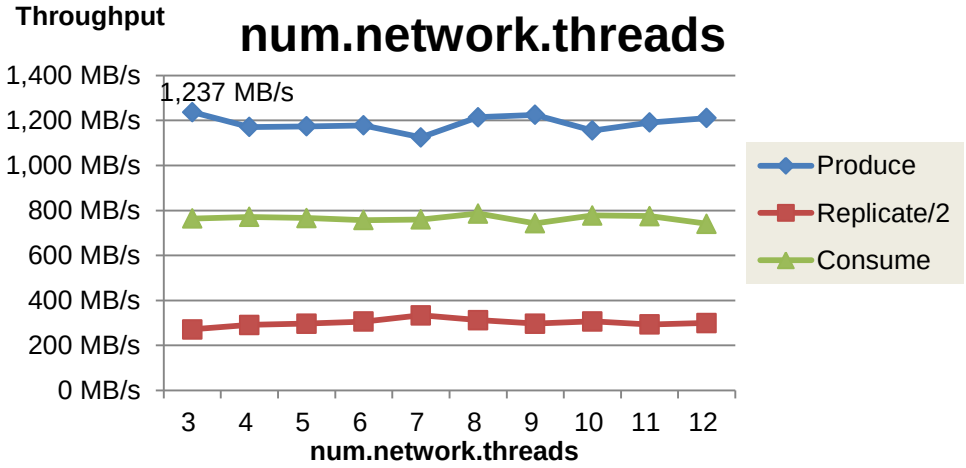


- gzipは圧縮率が高い分Byteレートは低く、snappyとlz4は圧縮率が低い分Byteレートは高い傾向
- ただしRecordレートを比べると圧縮なし(none)が最も高かった

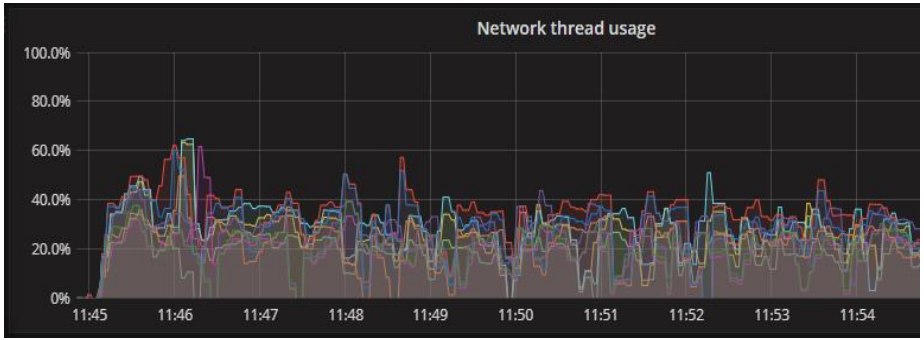
① Producerの送信性能のチューニング

HITACHI
Inspire the Next

- 受信側 Broker の Socket Server のネットワークスレッド数（ num.network.threads ）を増やして測定



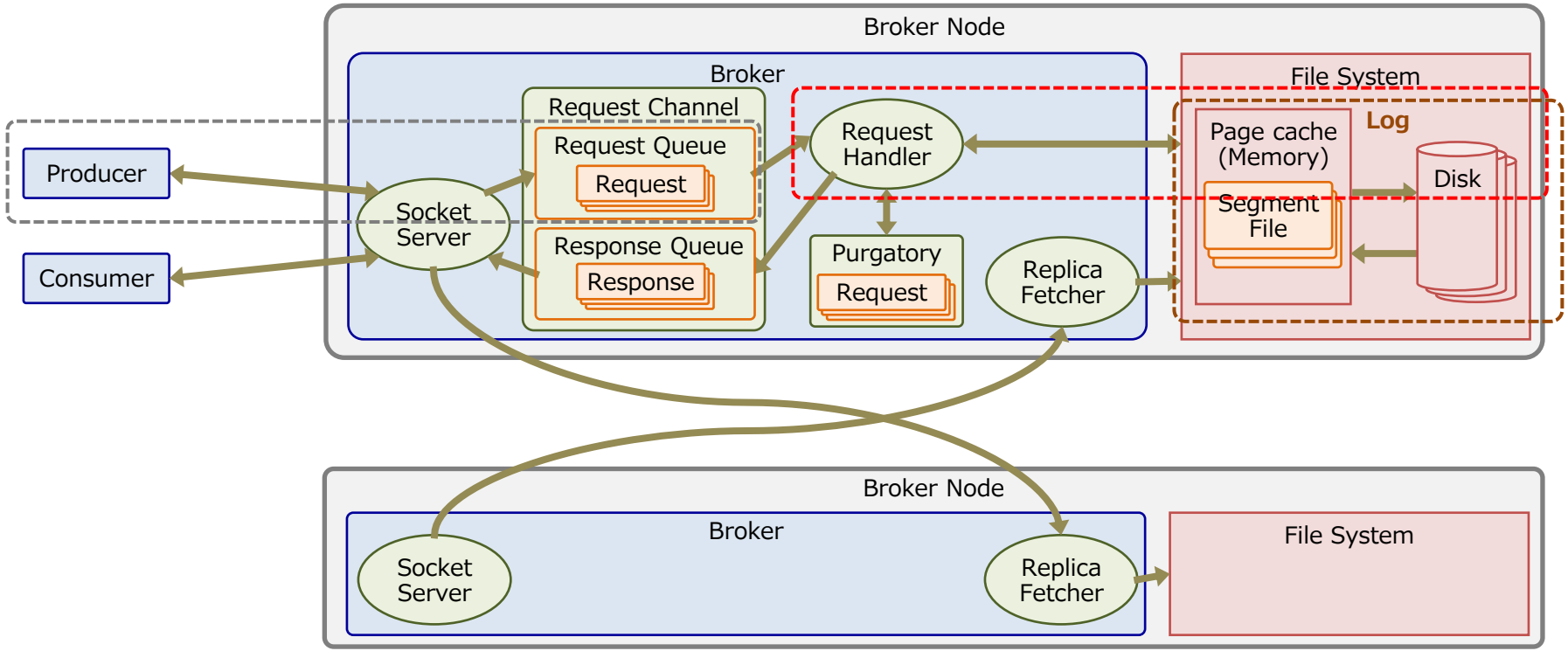
ネットワークスレッドのCPU使用率
（3 num.network.threads × 3 broker =9 threads ）



- ネットワークスレッド数を増やしてもProduceスループットはほぼ横ばいであった
- ネットワークスレッドのCPU使用率は40%程度のためボトルネックではない

② BrokerのLogフラッシュ性能のチューニング

- **目的：** acks=1 で Produceスループットを最大化する
  - ➢ Logフラッシュまで含めた性能を最適化する

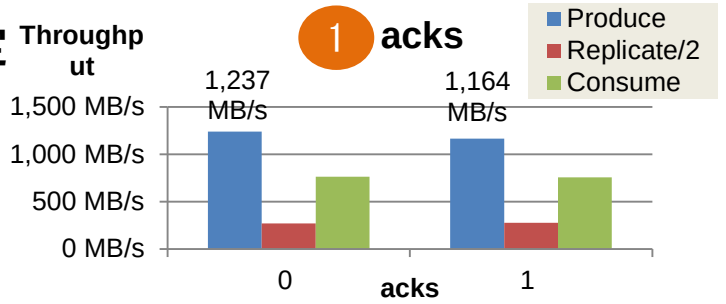


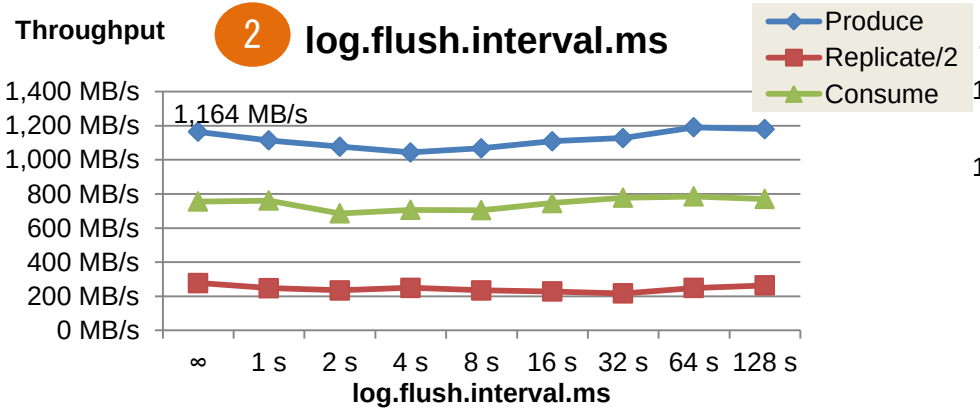
② BrokerのLogフラッシュ性能のチューニング

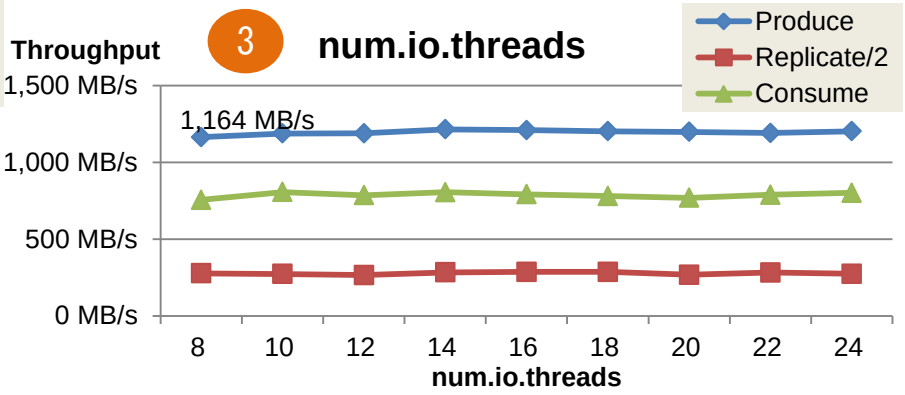
HITACHI
Inspire the Next

① Logフラッシュを含む性能を測定するためacks=1に設定
② Logフラッシュ間隔をチューニング
- デフォルト設定ではOSのフラッシュに任せている

③ リクエスト処理スレッド数をチューニング





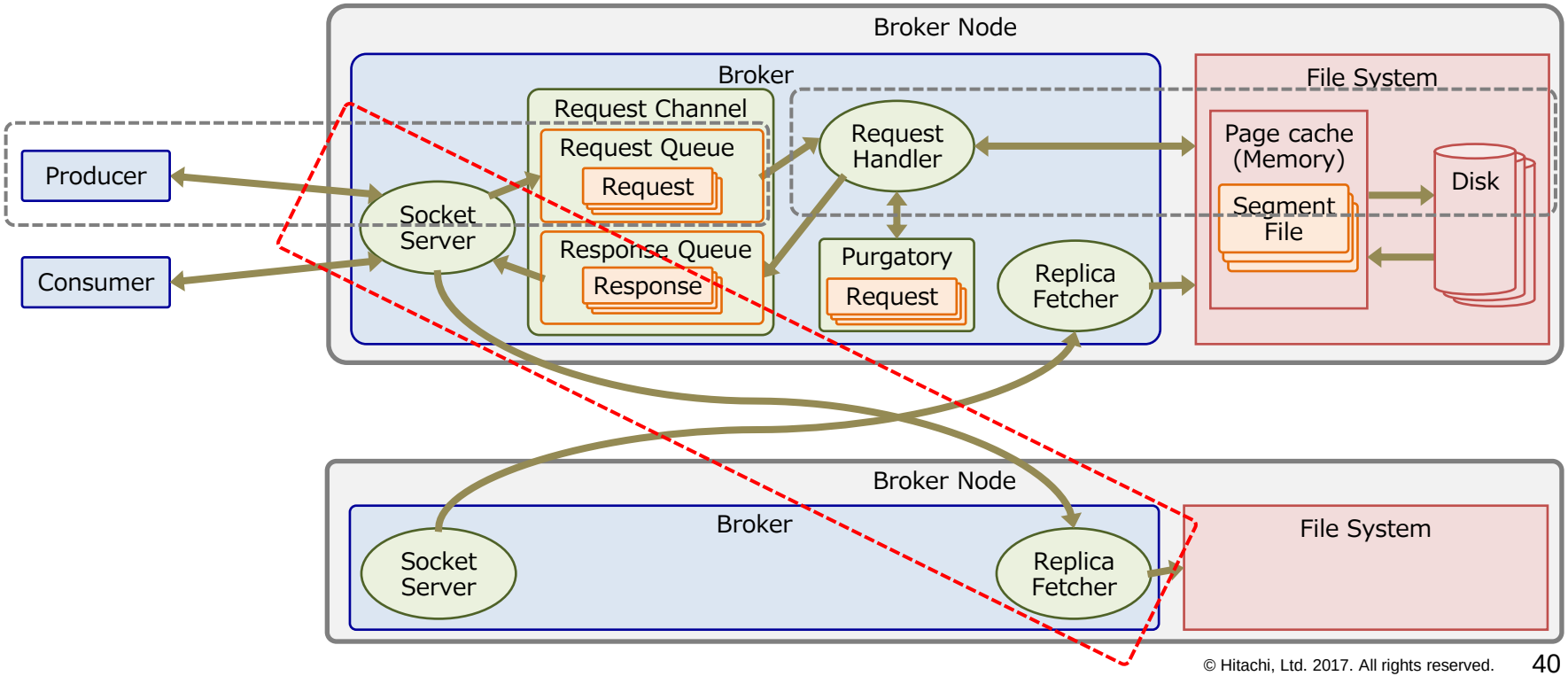


- acks=0->1でディスクI/Oがボトルネックとなり、Produceスループットは1,237 -> 1,164MB/sに低下
- log.flush.interval.ms / num.io.threads のチューニングはほぼ効果がなかった

③ Broker間のレプリケーション性能のチューニング

HITACHI
Inspire the Next

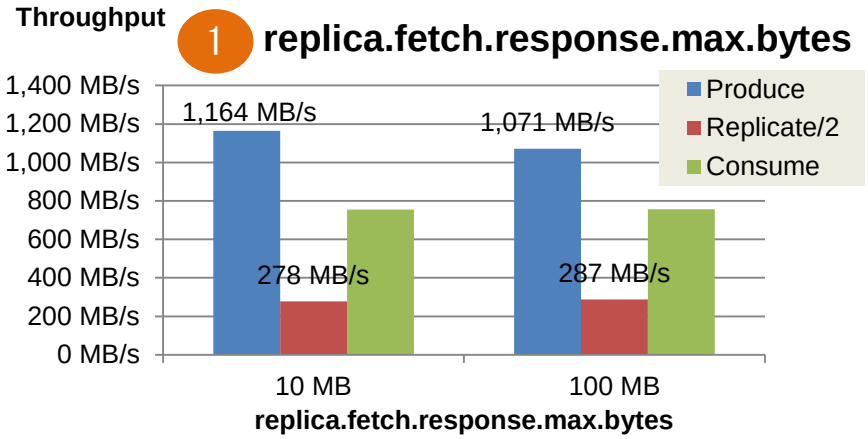
- **目的：** acks=all で Replicateスループットを最大化する
  - Produce/Replicateスループットの変化を見るため、まずはacks=1でチューニングを行い、最後にacks=allに変更する

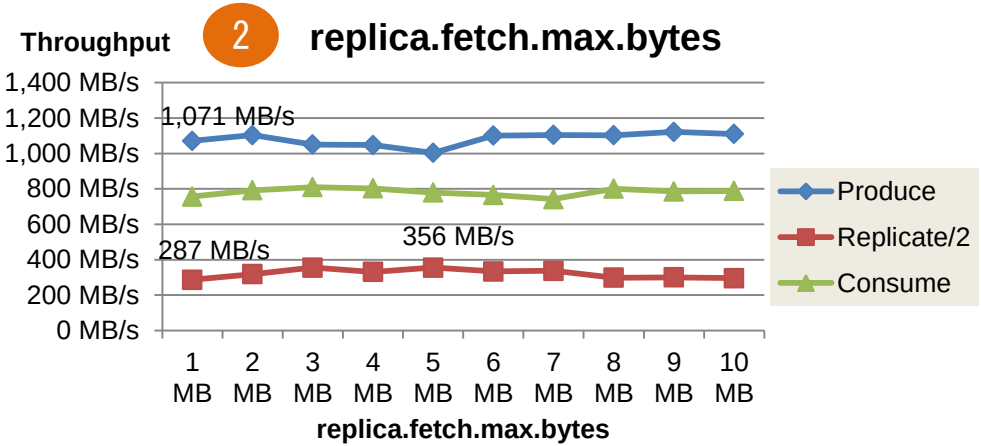


③ Broker間のレプリケーション性能のチューニング

HITACHI
Inspire the Next

- Replicateスループットを最大化するため、Replica fetcherが送信するFetchリクエストの設定をチューニング
  - ① Fetchリクエストで取得する最大データサイズを変更
  - ② Fetchリクエストで取得するPartition単位の最大データサイズを増やす



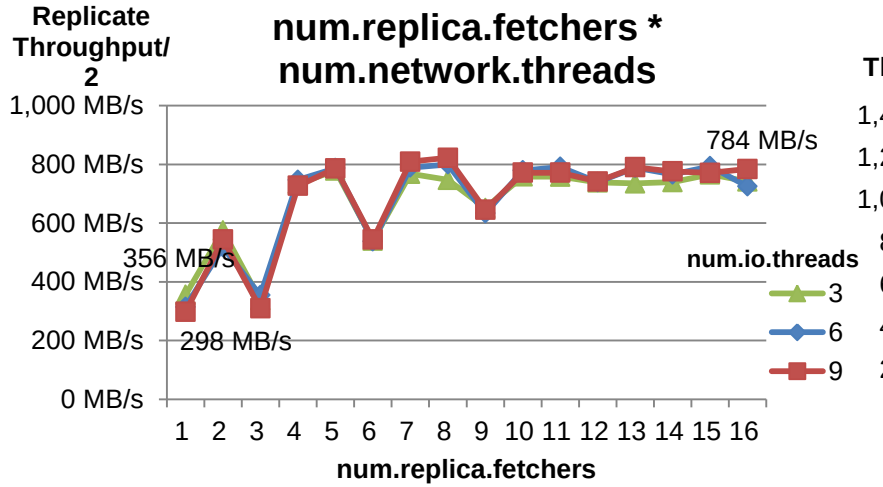


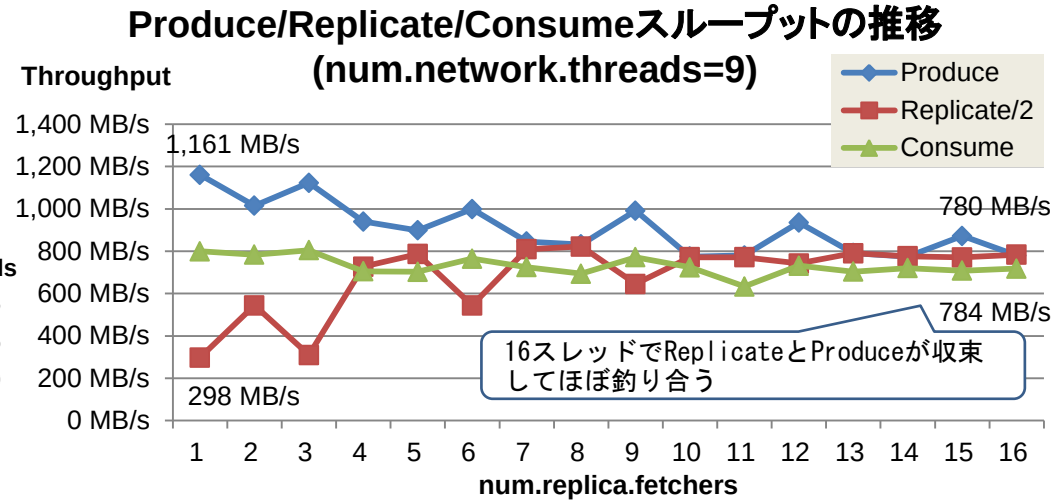
- replica.fetch.response.max.bytes=10->100MBでReplicateスループットは 278->287 MB/sに若干向上
- replica.fetch.max.bytes=1->5MBでReplicateスループットは 287->356 MB/sに向上

③ Broker間のレプリケーション性能のチューニング

HITACHI
Inspire the Next

- Replica fetcherのスレッド数と、Socket Serverのスレッド数を同時にチューニング



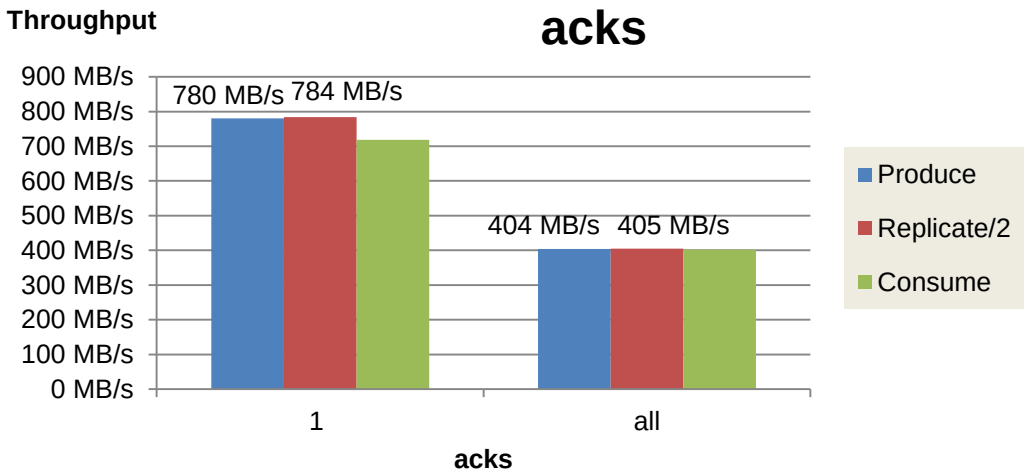


- num.replica.fetchersのチューニングは効果が大きいが、num.io.threadsの影響は少なかった
- num.replica.fetchersがBrokerの倍数(3の倍数)の時は、一部のReplica Fetcherにすべて自BrokerのPartitionを割り当ててしまうため、そのReplica fetcherが使用されず性能が低下
- num.io.threads=3->9, num.replica.fetchers=1->16でReplicateスループットは359->784MB/sに向上

③ Broker間のレプリケーション性能のチューニング

HITACHI
Inspire the Next

- acks=1 -> all に変更して、レプリケーションを非同期 -> 同期に変更

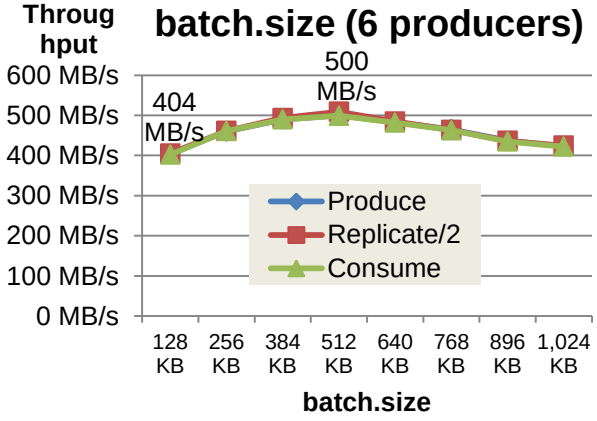


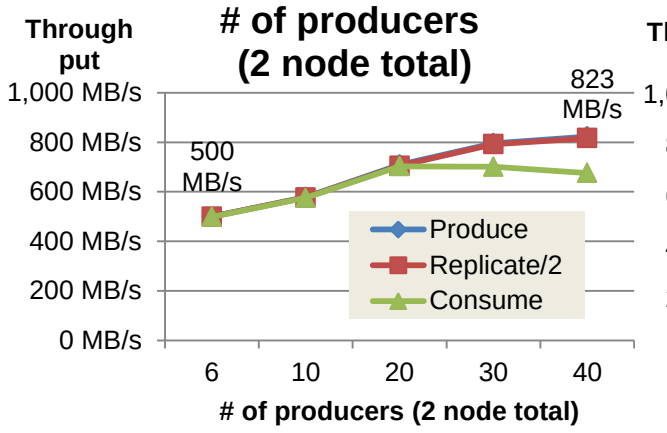
acks=1の時点でProduceとReplicateのスループットがほぼ釣り合っていたにも関わらず、
acks=allに設定するとスループットは大幅に低下してしまった

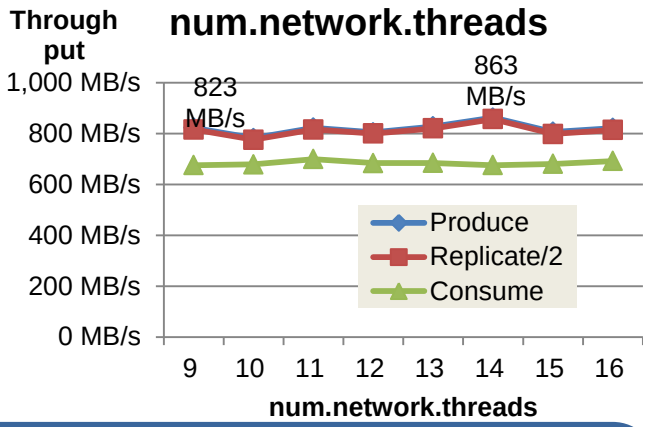
Producer性能の再チューニング

HITACHI
Inspire the Next

- acks=allに設定してProducer送信性能を再チューニング





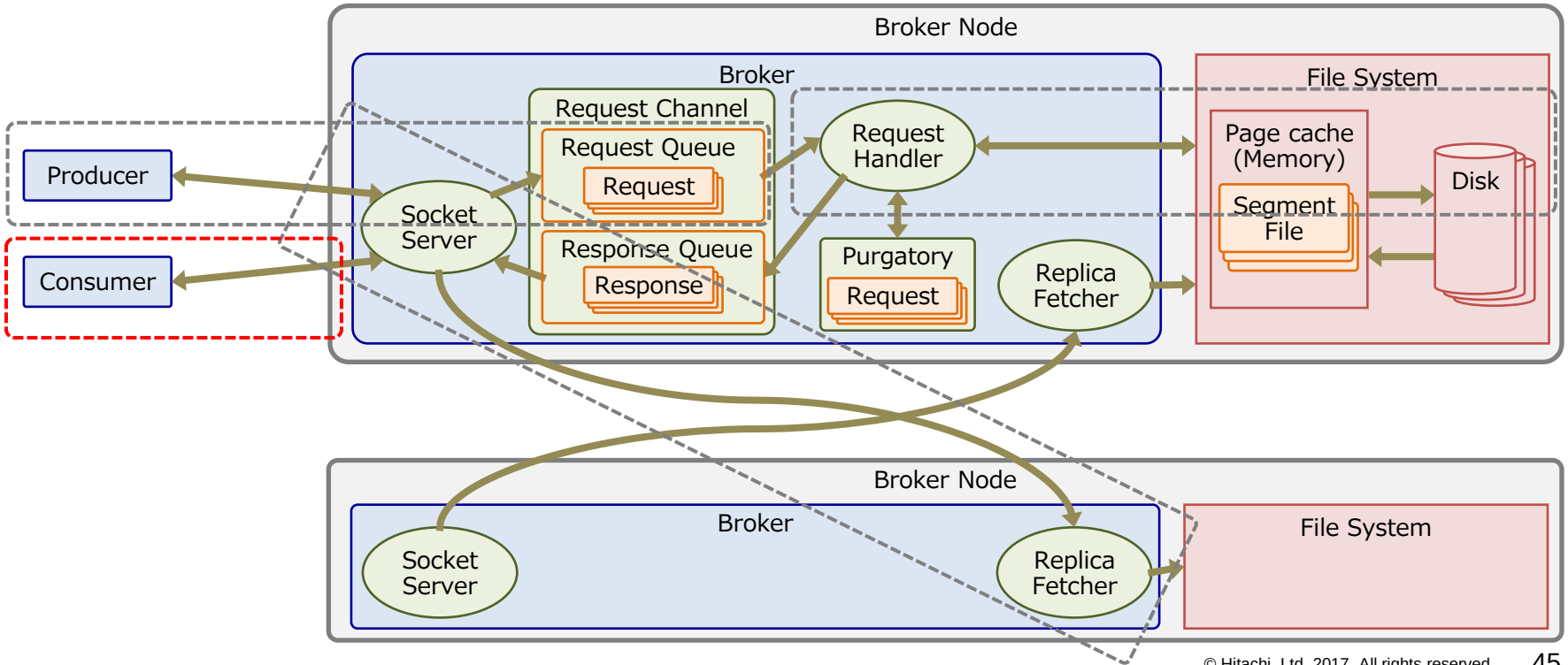


- batch.size= 128->512KBで、Produceスループットは 404->500 MB/s に向上
  - acks=allではRecord Batchサイズhを増やして1回のリクエストサイズを大きくした方が有利？
- Producer数= 6->40で、Produceスループットは 500->823 MB/s に向上
  - 1Producerあたりユーザ/ネットワークスレッドで2コア使用するため、最大40Producer(80コア)
  - acks=allではProducer数(=Brokerとのコネクション数)を増やすとよい？
- num.network.threads= 3->14で、Produceスループットは 823->863 MB/s に若干向上

④ Consumerの取得性能のチューニング

HITACHI
Inspire the Next

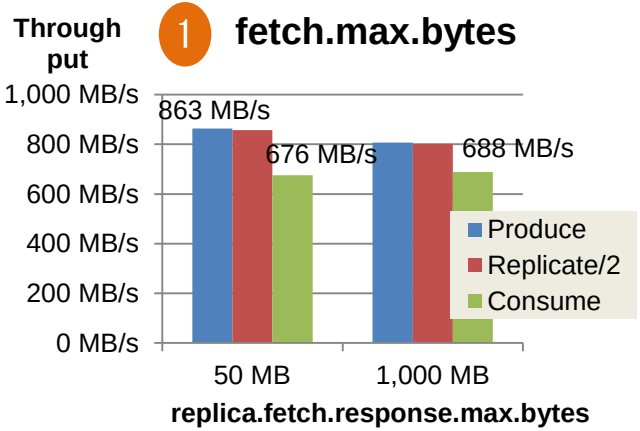
- **目的：** acks=all で Consumeスループットを最大化する
  - ➢ Producerの送信からConsumerの受信まで、システム全体を通したスループットを最適化する

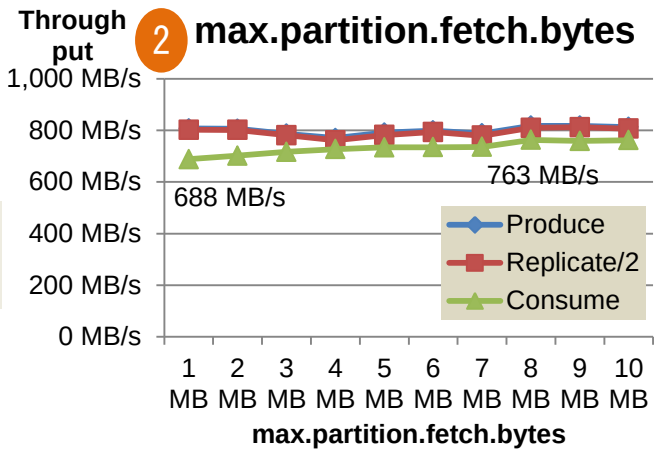


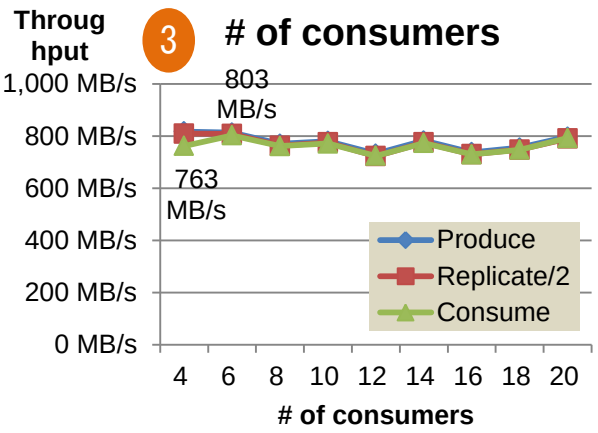
④ Consumerの取得性能のチューニング

HITACHI
Inspire the Next

- Fetchリクエストの、①合計取得サイズと②Partition単位の最大取得サイズを変更
- ③Consumer GroupのConsumer数を変更







- ◆ fetch.max.bytes= 50->1000MBで、Consumeスループットは 676->688 MB/s に若干向上
- ◆ max.partition.fetch.bytes= 1->8MBで、 Consumeスループットは 688->763 MB/s に向上
- ◆ Consumer数=4->6で、 Consumeスループットは 763->803 MB/s に若干向上

⇒ Produce/Replicate/Consumeスループットがほぼ一致し、システム全体で 803 MB/sを達成

システム全体のレイテンシは？

HITACHI
Inspire the Next

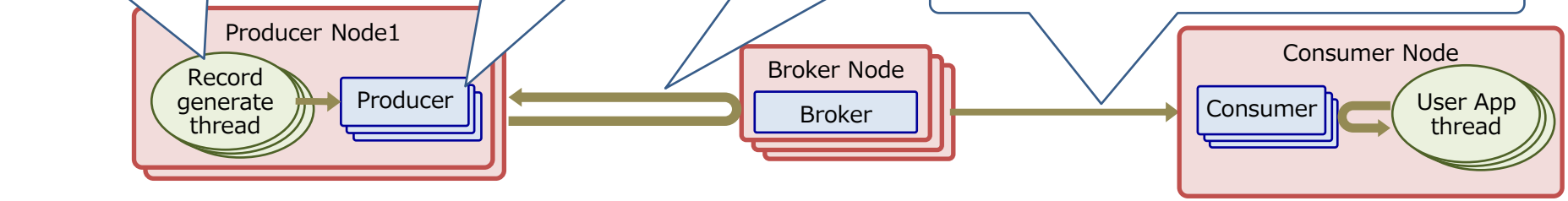
Producerに追加したレコードをConsumerで取得するまでの時間を測定

② Producerのバッファリング時間: 50s
- 1Producerの送信レートは 803MB/s ÷ 40 Producer≒ 20MB/s
- 1Producerのbuffer.memoryは 1024MB
- よって、レコード追加速度 ＞ Producerの送信速度 の場合は バッファが満杯となり、1024 MB ÷ 20MB ≒ 50s

① Send APIのブロック時間: 100ms
Producerのバッファ空き待ち

③ Produceリクエストのレイテンシ: 2.5s
acks=allなのでレプリケーションを含む

④ Fetchリクエストのレイテンシ: 150ms



- レコード追加速度がProducer送信性能を超えてバッファが詰まるため、全体のレイテンシは 約53秒
- レコード追加速度をKafkaの最大スループット(800MB)以下に抑えれば、レイテンシは 3秒程度 と予想
  - ➢ レコード蓄積を待つためにlinger.msを増やす必要があるかもしれない

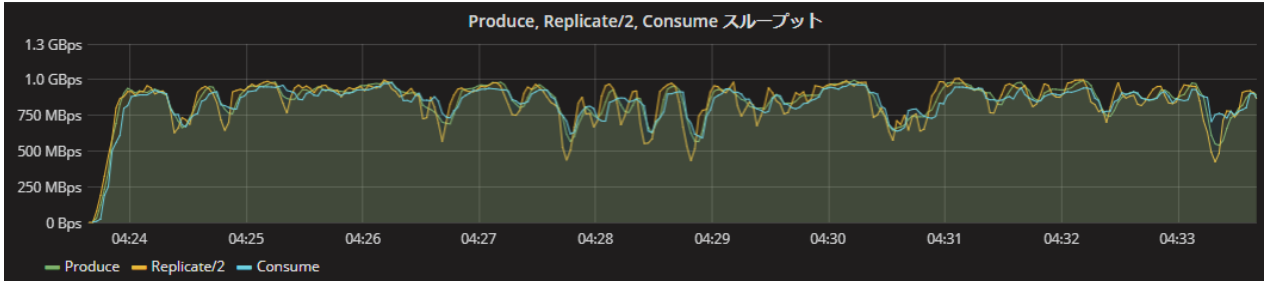
HITACHI
Inspire the Next

5. まとめ

まとめ

HITACHI
Inspire the Next

- 3レプリカの同期レプリケーション（acks=all）で平均 803MB/sのスループットを達成
  - ネットワーク帯域の理論値（1,170MB/s）の約70%
  - スループットは時間とともに上下しており、1,000MB/s（理論値の85%）程度の帯域は使用している



- チューニングのポイント
  - Producer
    - batch.size と Producer数（=Brokerとのコネクション数）を増やす
  - Broker
    - num.replica.fetchers を増やす
      （ただしReplica fetcherに対するPartition割り当ての偏りに注意）
  - Consumer
    - FetchサイズとConsumer GroupのConsumer数を増やす

他社商品名、商標等の引用に関する表示

HITACHI
Inspire the Next

- Apache Kafka, Apache ZooKeeperは、Apache Software Foundationの米国およびその他の国における登録商標または商標です。
- Javaは、Oracle Corporation及びその子会社、関連会社の米国およびその他の国における登録商標です。
- HITACHIは、株式会社 日立製作所の商標または登録商標です。
- その他記載の会社名、製品名などは、それぞれの会社の商標もしくは登録商標です。