Apache Kafka

Why Kafka?

When we have....
Aren't they **Good?**

They all are **Good**
But **not** for **all use-cases.**

Use Cases

- General purpose Message Bus

- Real-Time Stream Processing

- Collecting User Activity Data

- Collecting Operational Metrics from applications, servers or devices

- Log Aggregation

- Change Data Capture

- Commit Log for distributed systems

What is Common?

- **Scalable**: Need to be Highly Scalable. A <u>lot</u> of Data. It can be billions of message.

- **Reliability** of messages, What If, I loose a small no. of messages. Is it **fine** with me ?.

- **Distributed :** Multiple Producers, Multiple Consumers

- **High-throughput**: Does not require to have **JMS** Standards, as it may be overkill for some use-cases like transportation of logs.
  - As per JMS, each message has to be **acknowledged** back.
  - Exactly one delivery guarantee requires **two-phase commit**.

Introduction

- Kafka is a Distributed Streaming Platform
- An Apache Project, initially developed by LinkedIn's SNA team
- Open sourced in 2011
- Written in Scala
- **Key Capabilities** are
  - Publish and subscribe to streams of records
    (similar to a message queue or enterprise messaging system)
  - Store streams of records in a fault-tolerant durable way
  - Process streams of records as they occur
- **Generally used for**
  - Building real-time streaming data pipelines that reliably get data between systems or apps
  - Building real-time streaming applications that transform or react to the streams of data
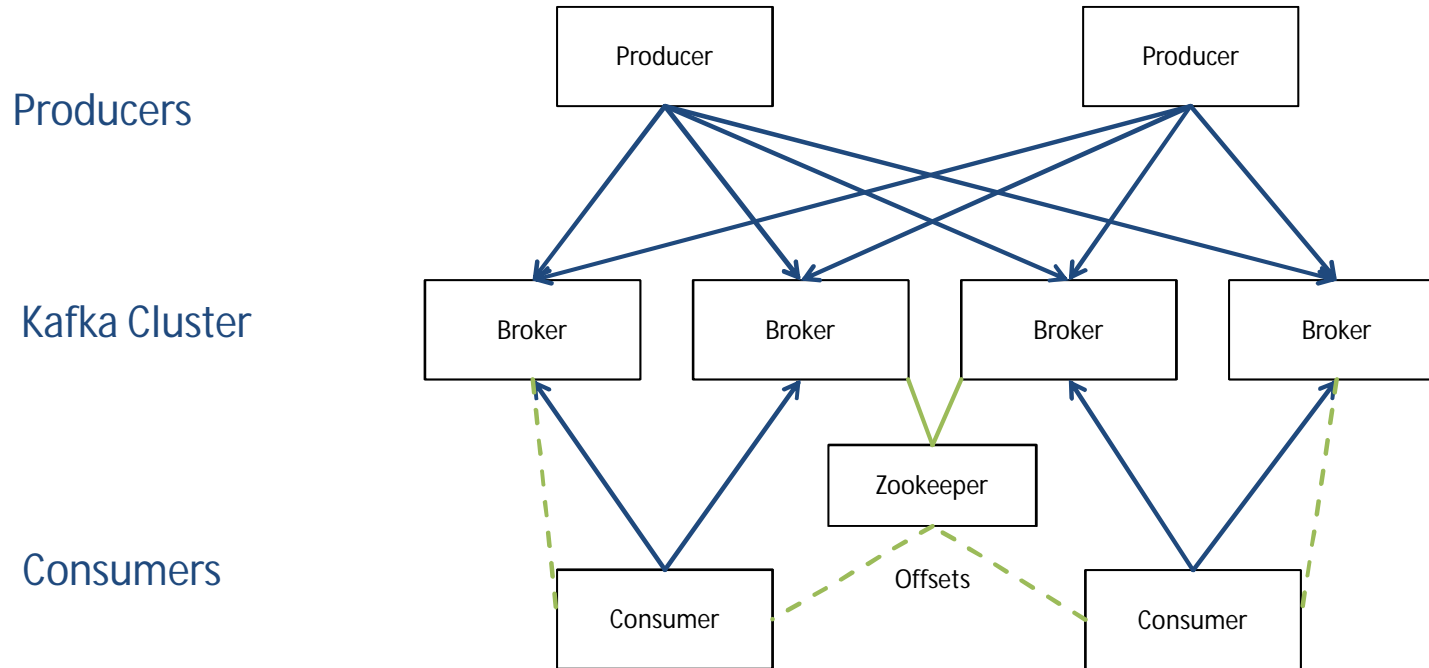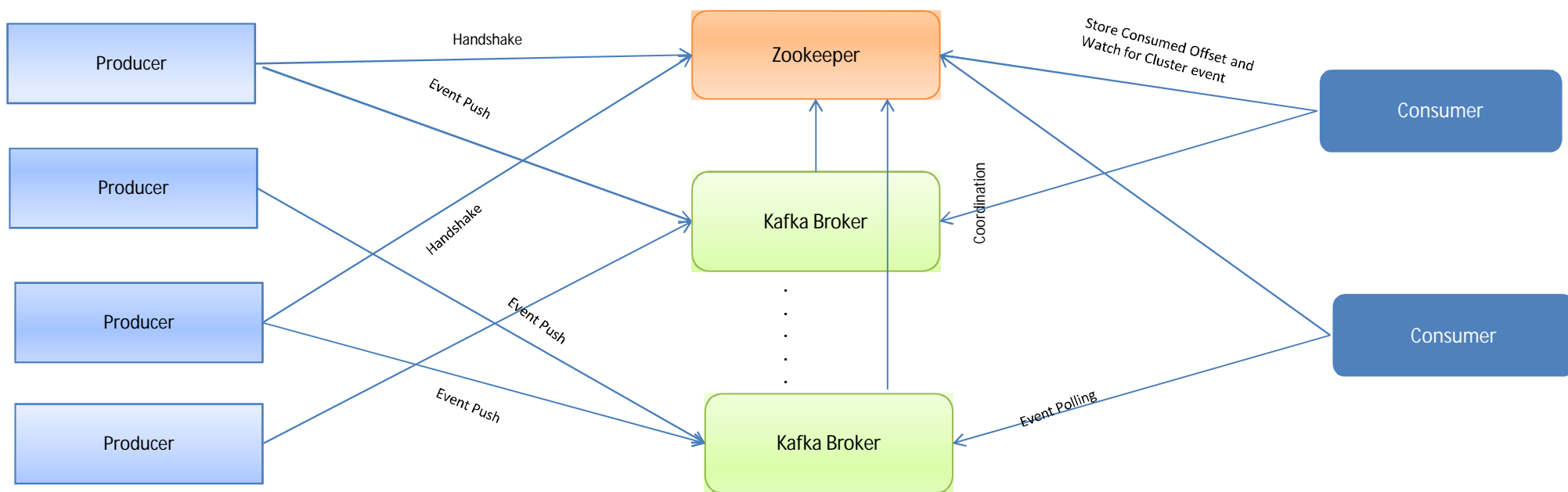
Overview
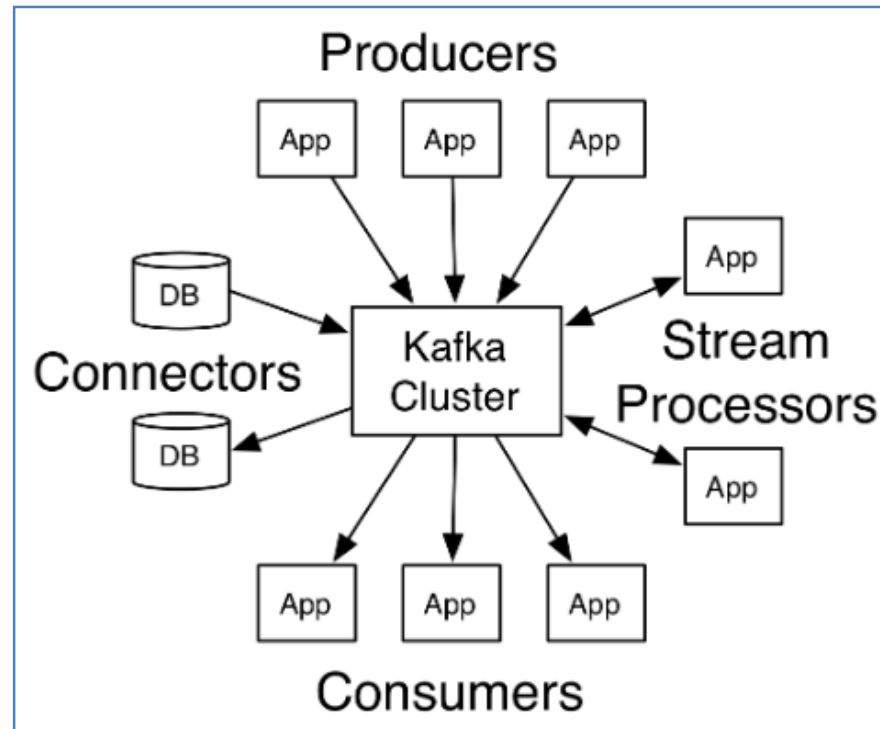
Key terminology

- Kafka maintains stream of records (messages) in categories called *topics*.
- Each record consists of a key, a value, and a timestamp.
- Processes that publish messages to a Kafka topic are called *producers*.
- Processes that subscribe to topics and process the feed of published messages are called *consumers*.
- Kafka is run as a cluster comprised of one or more servers each of which is called a *broker*.
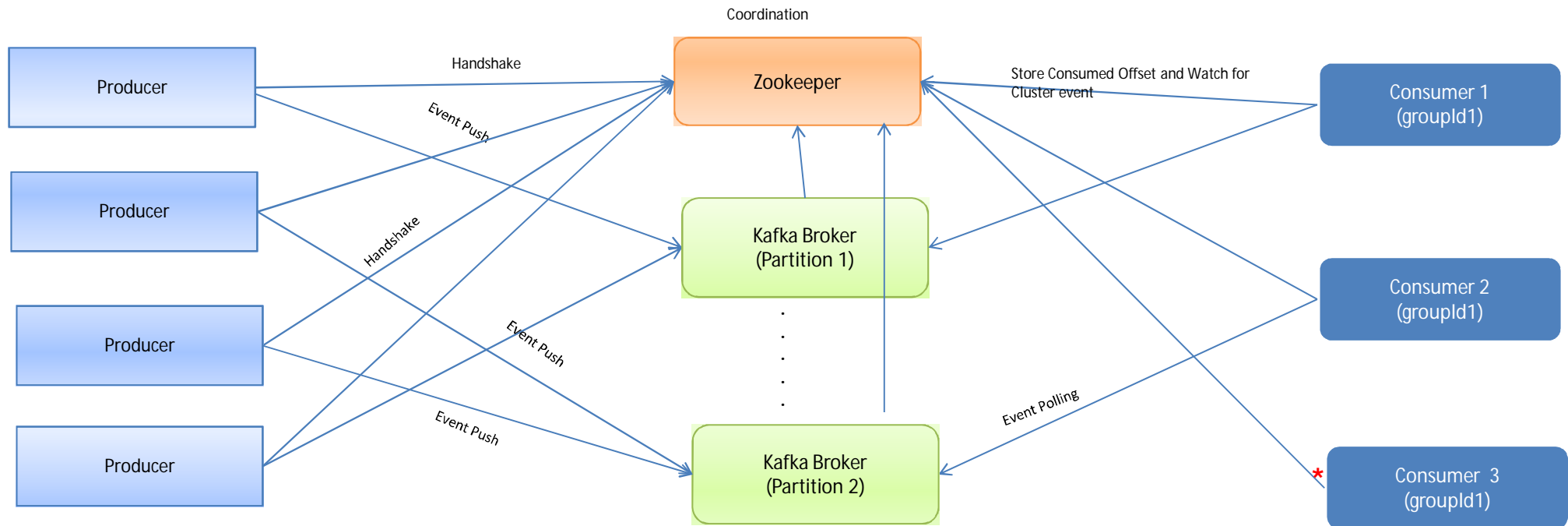- Communication between all components is done via a high performance simple binary API over TCP protocol

Architecture

Producers

Kafka Cluster

Consumers

How it works

How it works (Queue)

Coordination

Zookeeper

Producer

Producer

Producer

Producer

Handshake

Event Push

Handshake

Event Push

Event Push

Kafka Broker (Partition 1)

.
.
.
.
.

Kafka Broker (Partition 2)

Event Polling

Store Consumed Offset and Watch for Cluster event

Consumer 1 (groupId1)

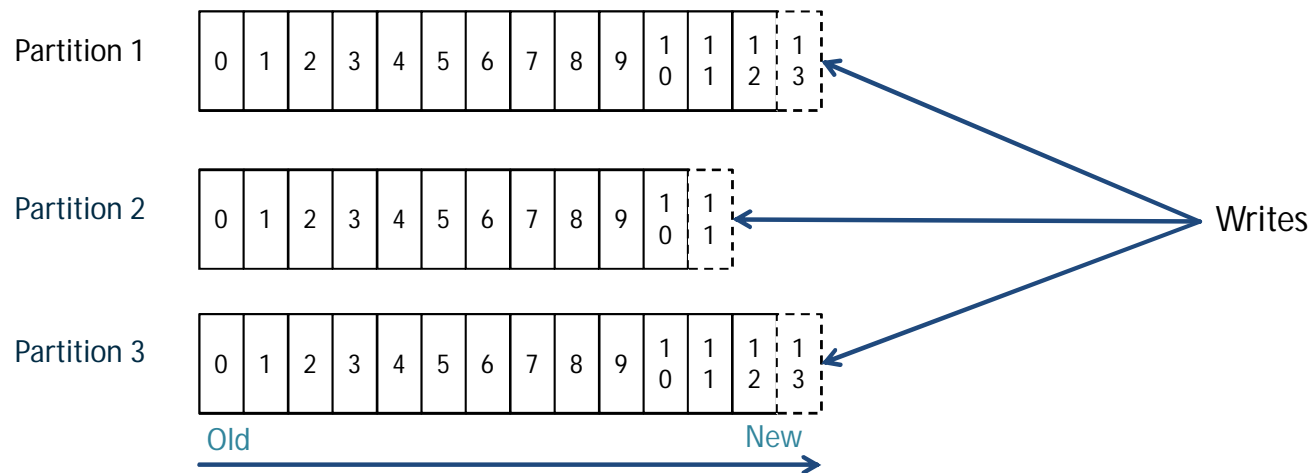**Consumer 2 (groupId1)**

Consumer 3 (groupId1)

*

* Consumer 3 would not receive any data, as number of consumers are more than number of partitions.

Design Elements

- Filesystem Cache
- Zero-copy transfer of messages
- Batching of Messages
- Batch Compression
- Automatic Producer Load balancing.
- Broker does not **Push** messages to Consumer, Consumer **Polls** messages from Broker.
- And Some others.
  - Cluster formation of Broker/Consumer using Zookeeper, So on the fly more consumer, broker can be introduced. The new cluster rebalancing will be taken care by Zookeeper
  - Data is persisted in broker and is not removed on consumption (till retention period), so if one consumer fails while consuming, same message can be re-consume again later from broker.
  - Simplified storage mechanism for message, not for each message per consumer.

Topics - Partitions

- Topics are broken up into ordered commit logs called **partitions**.
- Each message in a partition is assigned a sequential id called an **offset**.
- Each partition is segregated in multiple **segments** to ease finding
- Each segment is composed of three files (log, index and timeindex)
- Data is retained for a configurable period of time*

Partition 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13

Partition 2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11

Partition 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13

Writes

Old → New

Message Ordering

- Ordering is only guaranteed within a partition for a topic
- To ensure ordering:
  - Group messages in a partition by key (producer)
  - Configure exactly one consumer instance per partition within a consumer group

Guarantees

- Messages sent by a producer to a particular topic partition will be appended in the order they are sent

- A consumer instance sees messages in the order they are stored in the log

- For a topic with replication factor N, Kafka can tolerate up to N-1 server failures without "losing" any messages committed to the log

Topics - Replication

- Topics can (and should) be replicated.
- The unit of replication is the partition
- Each partition in a topic has 1 leader and 0 or more replicas.
- A replica is deemed to be "in-sync" if
  - The replica can communicate with Zookeeper
  - The replica is not "too far" behind the leader (configurable)
- The group of in-sync replicas for a partition is called the *ISR* (In-Sync Replicas)
- The Replication factor cannot be lowered

Topics - Replication

- Durability can be configured with the producer configuration *request.required.acks*
  - *0* The producer never waits for an ack
  - *1* The producer gets an ack after the leader replica has received the data
  - *-1* The producer gets an ack after all ISRs receive the data
- Minimum available ISR can also be configured such that an error is returned if enough replicas are not available to replicate data

Durable Writes

- Producers can choose to *trade* throughput for durability of writes:

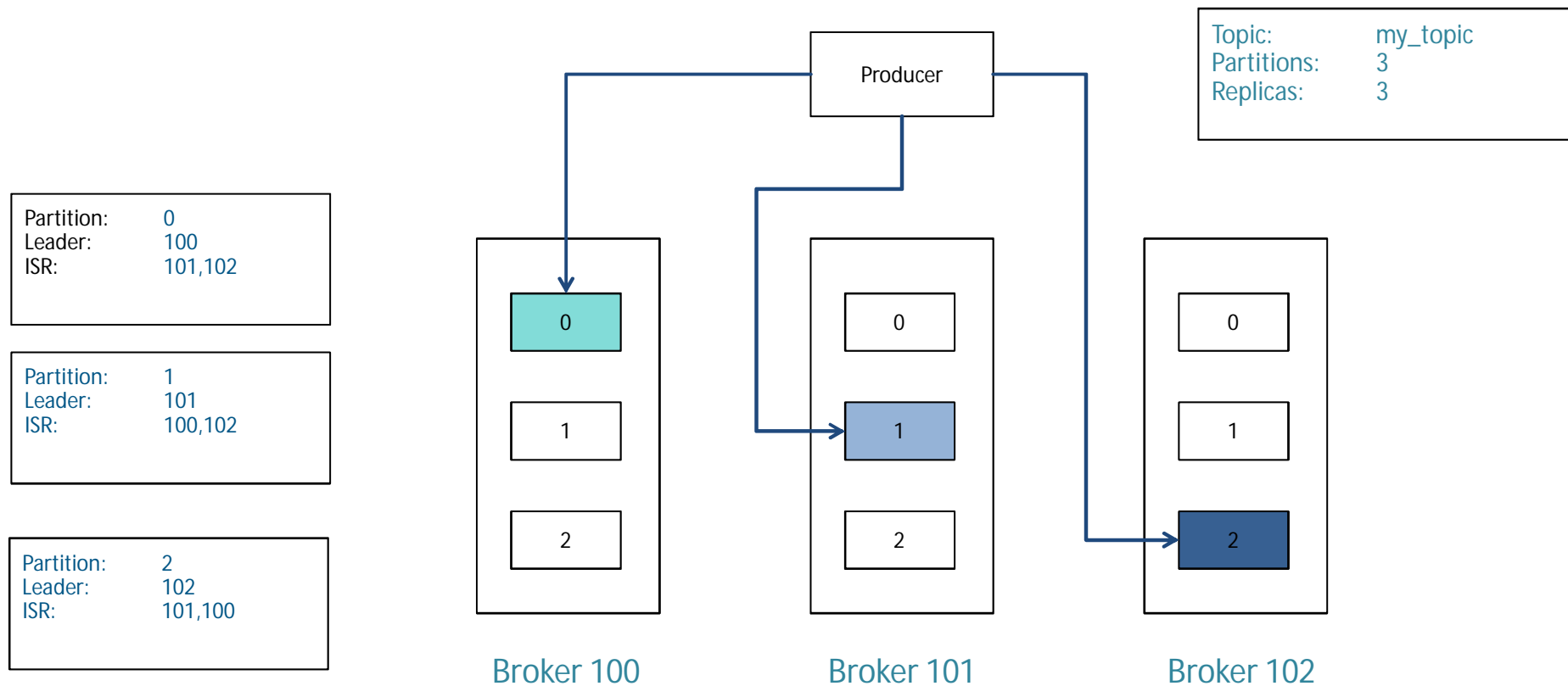| Durability	Behaviour	Per Event Latency	Required Acknowledgements (request.required.acks)
Highest	ACK all ISRs have received	Highest	-1
Medium	ACK once the leader has received	Medium	1
Lowest	No ACKs required	Lowest	0

- Throughput can also be raised with ***more brokers***… (so do this instead)!

- A sane configuration:

Property	Value
replication	3
min.insync.replicas	2
request.required.acks	-1

Producer

- Producers publish to a topic of their choosing (push)
- Load can be distributed
  - Typically by "round-robin"
  - Can also do "semantic partitioning" based on a key in the message
- Brokers load balance by partition
- Can support async (less durable) sending
- All nodes can answer metadata requests about:
  - Which servers are alive
  - Where leaders are for the partitions of a topic
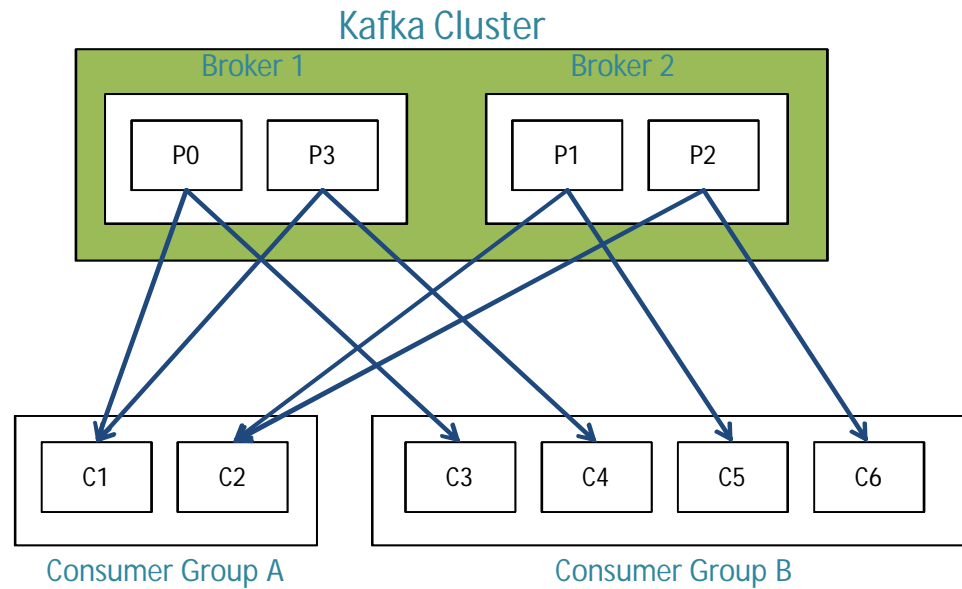
Producer – Load Balancing and ISRs

Topic: my_topic
Partitions: 3
Replicas: 3

Producer

Partition: 0
Leader: 100
ISR: 101,102

Partition: 1
Leader: 101
ISR: 100,102

Partition: 2
Leader: 102
ISR: 101,100

| 0 |
| 1 |
| 2 |

Broker 100

| 0 |
| 1 |
| 2 |

Broker 101

| 0 |
| 1 |
| 2 |

Broker 102

Consumer - Groups

- Consumers can be organized into Consumer Groups
- Consumer Group is a logical group of consumers
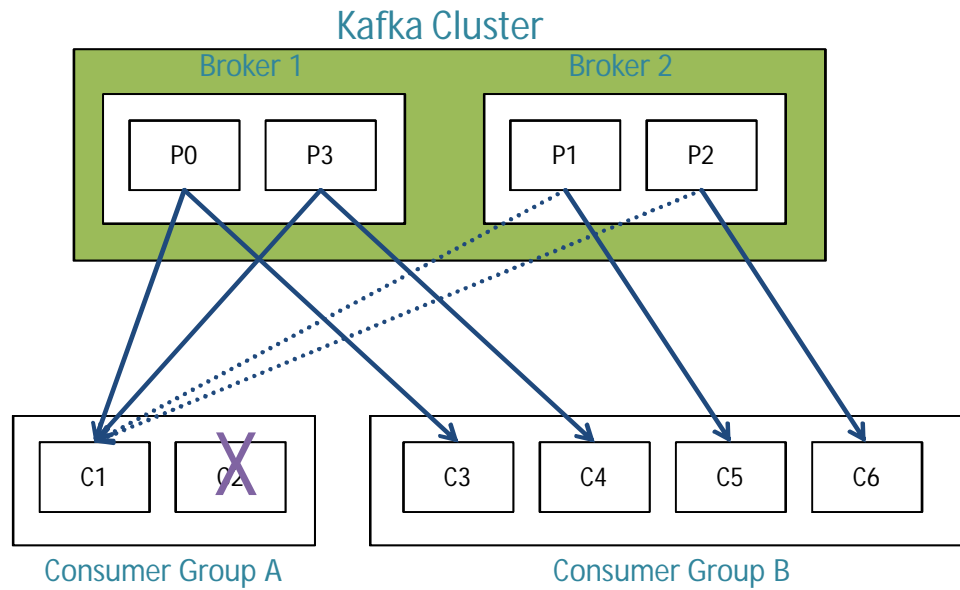
Common Patterns:

- 1) All consumer instances in one group
  - Acts like a traditional queue with load balancing
- 2) All consumer instances in different groups
  - All messages are broadcast to all consumer instances
- 3) "Logical Subscriber" – Many consumer instances in a group
  - Consumers are added for scalability and fault tolerance
  - Each consumer instance reads from one or more partitions for a topic
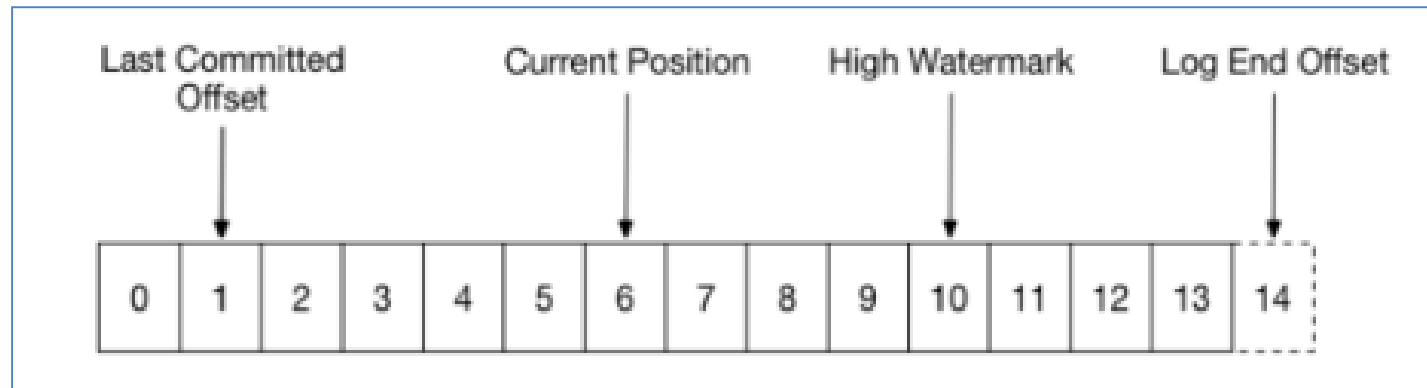 - There cannot be more consumer instances than partitions

Consumer - Groups

Kafka Cluster

Broker 1 · Broker 2

P0 · P3 · P1 · P2

Consumer Groups provide isolation to topics and partitions

C1 · C2 · C3 · C4 · C5 · C6

Consumer Group A · Consumer Group B

Consumer - Groups

Kafka Cluster

Broker 1

Broker 2

P0   P3

P1   P2

Can rebalance
themselves

C1   C2

Consumer Group A

C3   C4   C5   C6

Consumer Group B

Consumer - Groups

- **Last Committed Offset** – is the offset of the last message consumed and acknowledged by the consumer
- **Current Position** – is the offset of the message that consumer is currently consuming
- **High Watermark** – is the offset of the last message that was successfully replicated to all partition
- **Log End Offset** - is the offset of the last message written to the partition

Delivery Semantics

- At least once   Default
  - ~~Messages are never lost but may be redelivered~~
- At most once
  - Messages are lost but never redelivered
- Exactly once
  - Messages are delivered once and only once

Delivery Semantics

- At least once

  – **Messages are never lost but may be redelivered**

- At most once

  – Messages are lost but never redelivered

- Exactly once

  – Messages are delivered once and only once
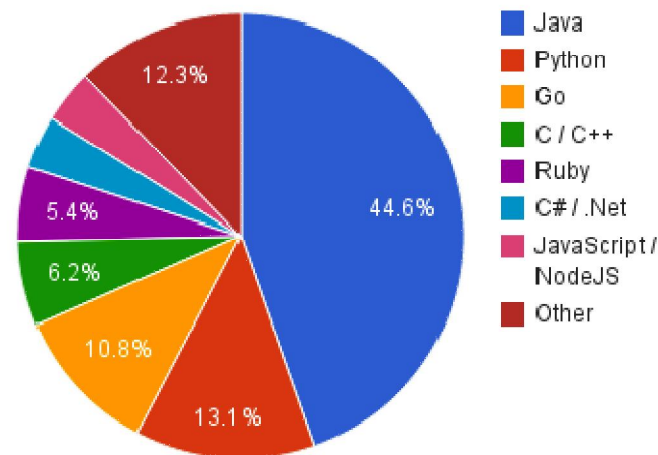
Much Harder
(Impossible??)

Getting Exactly Once Semantics

- Must consider two components
  - Durability guarantees when *publishing* a message
  - Durability guarantees when *consuming* a message
- Producer
  - What happens when a produce request was sent but a network error returned before an ack?
  - Use a single writer per partition and check the latest committed value after network errors
- Consumer
  - Include a unique ID (e.g. UUID) and de-duplicate.
  - Consider storing offsets with data

Kafka Clients

- Remember Kafka implements a binary TCP protocol.
- All clients except the JVM-based clients are maintained external to the code base.
- Full Client List here:

Kafka Producer/Consumer Language

Legend:
- Java — 44.6%
- Python — 13.1%
- Go — 10.8%
- C / C++ — 6.2%
- Ruby — 5.4%
- C# / .Net
- JavaScript / NodeJS
- Other — 12.3%

Java Producer Example – Old (< 0.8.1)

```
/* start the producer */
  private void start() {
  producer = new Producer<String, String]]>(config);
  }


  /* create record and send to Kafka
  * because the key is null, data will be sent to a random partition.
  * the producer will switch to a different random partition every 10 minutes
  **/
  private void produce(String s) {
  KeyedMessage<String, String]]> message = new KeyedMessage<String,
String]]>(topic, null, s);
  producer.send(message);
  }
```

Producer - New

```
    /**
     * Send the given record asynchronously and return a future which will eventually contain the
response information.
     *
     * @param record The record to send
     * @return A future which will eventually contain the response information
     */
    public Future send(ProducerRecord record);

    /**
     * Send a record and invoke the given callback when the record has been acknowledged by the
server
     */
    public Future send(ProducerRecord record, Callback callback);
}
```
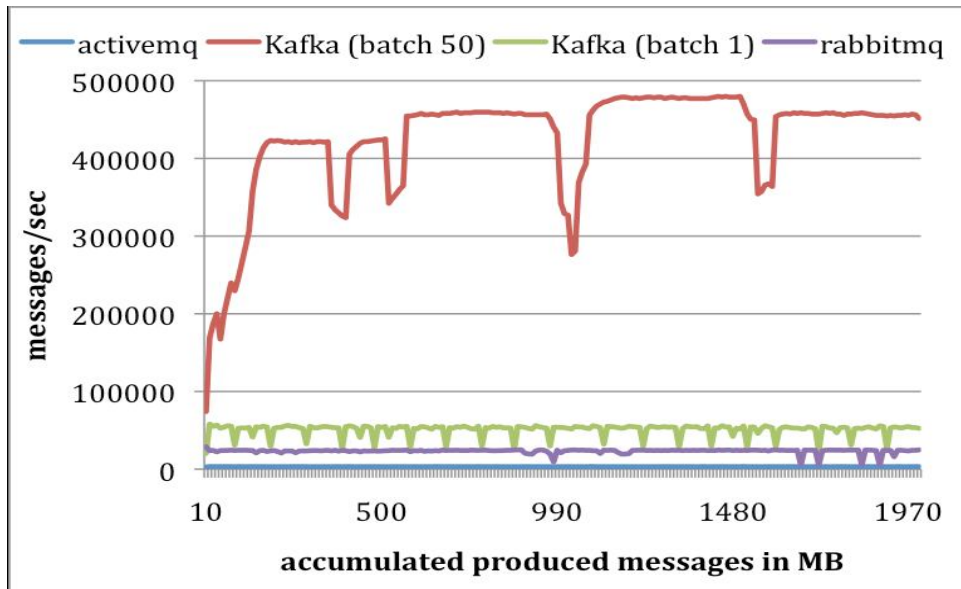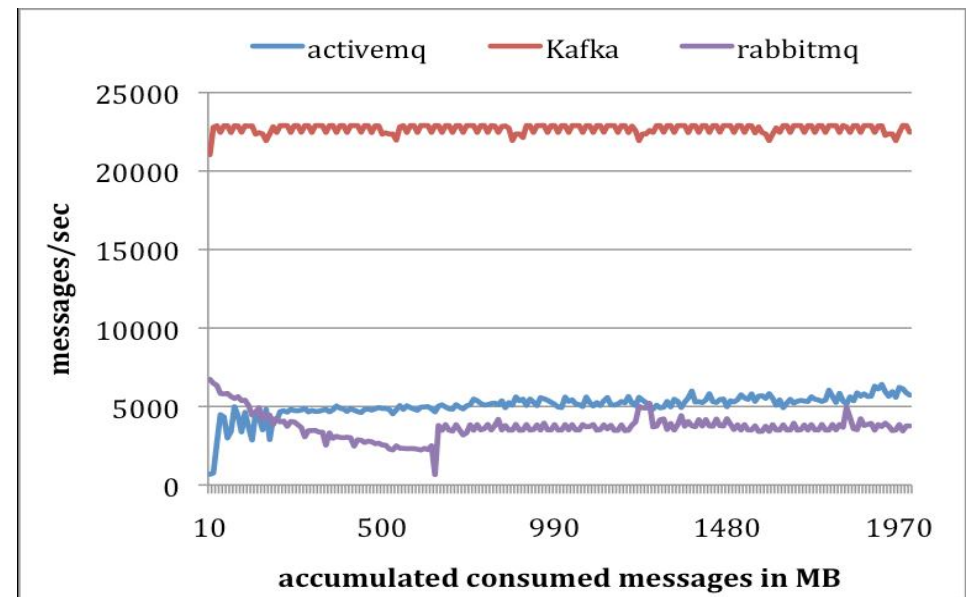
```java
// configure
Properties config = new Properties();
config.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
KafkaProducer producer = new KafkaProducer(config);

// create and send a record
ProducerRecord record = new ProducerRecord("topic", "key".getBytes(), "value".getBytes());
Future<RecordMetadata> response = producer.send(record); // this is always non-blocking
System.out.println("The offset was: " + response.get().offset()); // get() blocks
```

Performance Numbers

Producer Performance

Consumer Performance

Credit : http://research.microsoft.com/en-us/UM/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf

Powered By

- LinkedIn - Apache Kafka is used at LinkedIn for activity stream data and operational metrics. This powers various products like LinkedIn Newsfeed, LinkedIn Today in addition to our offline analytics systems like Hadoop.
- Tumblr - See this
- Mate1.com Inc. - Apache kafka is used at Mate1 as our main event bus that powers our news and activity feeds, automated review systems, and will soon power real time notifications and log distribution.
- Tagged - Apache Kafka drives our new pub sub system which delivers real-time events for users in our latest game - Deckadence. It will soon be used in a host of new use cases including group chat and back end stats and log collection.
- Boundary - Apache Kafka aggregates high-flow message streams into a unified distributed pubsub service, brokering the data for other internal systems as part of Boundary's real-time network analytics infrastructure.
- DataSift - Apache Kafka is used at DataSift as a collector of monitoring events and to track user's consumption of data streams in real time. http://highscalability.com/blog/2011/11/29/datasift-architecture-realtime-datamining-at-120000-tweets-p.html
- Wooga - We use Kafka to aggregate and process tracking data from all our facebook games (which are hosted at various providers) in a central location.
- AddThis - Apache Kafka is used at AddThis to collect events generated by our data network and broker that data to our analytics clusters and real-time web analytics platform.
- Urban Airship - At Urban Airship we use Kafka to buffer incoming data points from mobile devices for processing by our analytics infrastructure.
- Metamarkets - We use Kafka to collect realtime event data from clients, as well as our own internal service metrics, that feed our interactive analytics dashboards.
- SocialTwist - We use Kafka internally as part of our reliable email queueing system.
- Countandra - We use a hierarchical distributed counting engine, uses Kafka as a primary speedy interface as well as routing events for cascading counting
- FlyHajj.com - We use Kafka to collect all metrics and events generated by the users of the website.
- Twitter - As part of their Storm stream processing infrastructure, e.g. this.
- uSwitch - See this blog.
- InfoChimps - Kafka is part of the InfoChimps real-time data platform.
- uSwitch - See this blog.
- Visual Revenue - We use Kafka as a distributed queue in front of our web traffic stream processing infrastructure (Storm).
- Oolya - Kafka is used as the primary high speed message queue to power Storm and our real-time analytics/event ingestion pipelines.
- Foursquare - Kafka powers online to online messaging, and online to offline messaging at Foursquare. We integrate with monitoring, production systems, and our offline infrastructure, including hadoop.
- Datadog - Kafka brokers data to most systems in our metrics and events ingestion pipeline. Different modules contribute and consume data from it, for streaming CEP (homegrown), persistence (at different "temperatures" in Redis, ElasticSearch, Cassandra, S3), or batch analysis (Hadoop).
- VisualDNA We use Kafka 1. as an infrastructure that helps us bring continuously the tracking events from various datacenters into our central hadoop cluster for offline processing, 2. as a propagation path for data integration, 3. as a real-time platform for future inference and recommendation engines
- Sematext - in SPM (performance monitoring), Kafka is used for performance metrics collection and feeds SPM's in-memory data aggregation (OLAP cube creation) as well as our CEP/Alerts servers. In SA (search analytics) Kafka is used in search stream collection before being aggregated and persisted in HBase.
- Mozilla - Kafka will soon be replacing part of our current production system to collect performance and usage data from the end-users browser for projects like Telemetry, Test Pilot, etc. Downstream consumers usually persist to either HDFS or HBase.
- Wize Commerce - At Wize Commerce (previously, NexTag), Kafka is used as a distributed queue in front of Storm based processing for search index generation. We plan to also use it for collecting user generated data on our web tier, landing the data into various data sinks like Hadoop, HBase, etc.
- Quixey - At Quixey, The Search Engine for Apps, Kafka is an integral part of our eventing, logging and messaging infrastructure.
- LinkSmart - Kafka is used at LinkSmart as an event stream feeding Hadoop and custom real time systems.
- Simple - We use Kafka at Simple for log aggregation and to power our analytics infrastructure.
- LucidWorks Big Data - We use Kafka for syncing LucidWorks Search (Solr) with incoming data from Hadoop and also for sending LucidWorks Search logs back to Hadoop for analysis.
- Square We use Kafka as a bus to move all systems events through our various datacenters. This includes metrics, logs, custom events etc. On the consumer side, we output into Splunk, Graphite, Esper-like real-time alerting.

Credit: https://cwiki.apache.org/confluence/display/KAFKA/Powered+By

Comments (3) Hide Comments | Collapse All | Add Comment

Thank You