ISIT312 Big Data Management

A Short Introduction to Apache Kafka

Dr Guoxin Su and Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Apache Kafka

Outline

- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

Meet Apache Kafka

Apache Kafka is a distributed event streaming platform designed to handle large-scale real-time data streams

Kafka was developed by LinkedIn and later on passed as an open-sourced project to Apache

At the moment a commercial version is managed by Confluent

Apache Kafka supports high-throughput, fault-tolerance, scalability, and low-latency for various use cases, such as real-time data pipelines, stream processing, log aggregation, and more

Apache Kafka follows a publish-subscribe messaging model, where producers publish messages to topics, and consumers subscribe to those topics to receive and process the messages

Apache Kafka allows to publish and to subscribe to events, to store events for as long as it is needed and to process and analyse events

Apache Kafka is often described as a distributed commit log or as a distributed streaming platform

Meet Apache Kafka

Event streaming captures data in real-time from event sources like databases, sensors, mobile devices, cloud services, and software applications in the form of streams of events

Event streaming stores the event streams for later retrieval, manipulation, processing, and reacting to the event streams in real-time

Event streaming routes the event streams to different destination technologies for further processing

Applications of event streaming:

- processing of payments and financial transactions in real-time, such as in stock exchanges, banks, and insurance companies
- real time monitoring of vehicle traffic
- monitoring and analysis of sensor data from IoT devices or other equipment
- collecting and reacting to customer interactions in retail, hotel and travel industry through mobile applications
- monitoring of patients patients in hospital care and predicting changes to ensure timely treatment in emergencies

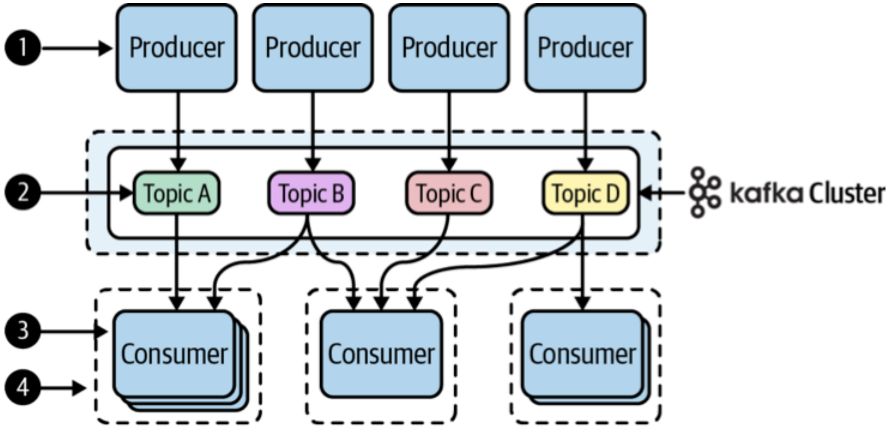
Apache Kafka

Outline

- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

Streaming Platform

Publish-subscribe pattern:



(1) Producers publish their data to one or more topics, without caring who comes along to read the data

(2) Topics are named streams (or channels) of related data

(3) Consumers are processes that read (or subscribe) to data in one or more topics

(4) Consumers can work together as a group (called a consumer group) in order to distribute work across multiple processes

Apache Kafka

Outline

- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

How Streams are Stored

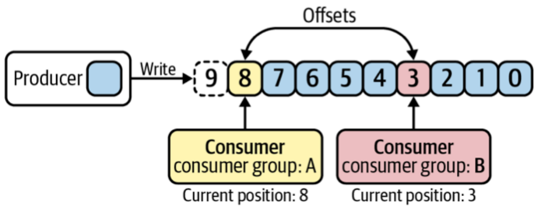
Commit Logs are append-only data structures that capture an ordered sequence of events

Each record is immutable; in order to model the update, append new records to the log

For example, a user purchases log (with 5 records):

User purchases log

```
timestamp=1597373669,user_id=1,purchases=1
timestamp=1597373669,user_id=2,purchases=1
timestamp=1597373669,user_id=3,purchases=1
timestamp=1597373669,user_id=4,purchases=1
timestamp=1597374265,user_id=1,purchases=2
```



Offset refers to a position of each record in a Kafka log

Offset allows consumers in the same group to maintain the position of the log they read

Apache Kafka

Outline

- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

Messages and Batches

A message is the smallest unit of data within Kafka (similar to a row or a record)

A messages is an array of bytes

A messages has a key which is a byte array

Keys are used when the messages are written into partitions

A batch is a collection of messages included in the same topic and partition

Due to the efficiency reasons messages are writen in batches

Batches are compressed to improve effciency of data transfers and storage at the costs

Grouping messages into batches is tradeoff between latency and throughput

If a batch is larger then more messages can be handled per unit of time, but it takes longer for an individual message to be propagated

Apache Kafka

Outline

- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

Schemas

A schema can be imposed on a message in order to provide interpretation a message

Apache Avro is used as a serialization framework originally developed for Hadoop

Avro provides a compact serialization format

Avro Avro schemas are defined using JSON, for example

Schema

```
{"namespace": "customerManagement.avro",
 "type": "record",
 "name": "Customer",
 "fields": [
             {"name": "id", "type": "int"},
             {"name": "name",  "type": "string"},
             {"name": "faxNumber", "type": ["null", "string"], "default": "null"}
           ]
}
```

Apache Kafka

Outline

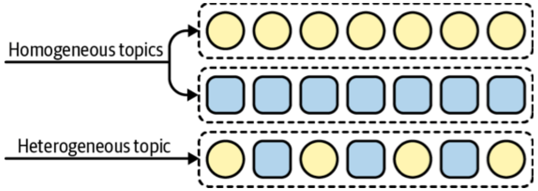
- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

Topics and Partitions

Messages are categorized into topics

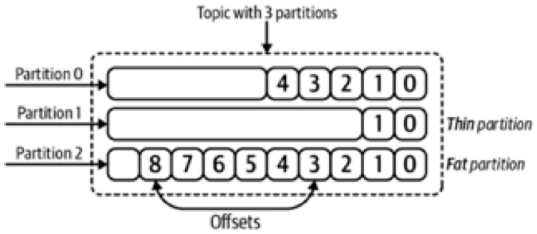
A topic is an event sequence

A topic can be homogeneous (of the same data type) or heterogeneous (of different data types)



A topic is similar to a relational table or file or records

Each topic can be distributed and comprised of multiple partitions



A partition is a single log located at one system in a cluster

Apache Kafka

Outline

- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

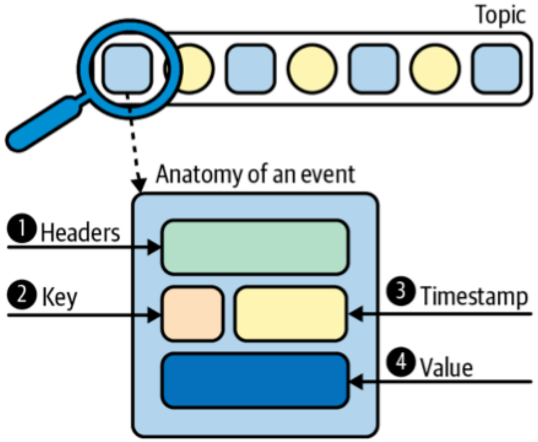
Events

A message is also called as an event

An event is a piece of data stored in a topic

An event refers to something that has happened (a fact),/li>

Anatomy of an event in Kafka:



Apache Kafka

Outline

- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

Producers and Consumers

There are two types of Kafka clients: producers and consumers

Producers create new messages; producers are also called as publishers or writers

A message from a producer is appended to a specific topic

Producer does not care what partition a specific message is written to

Consumers read messages; consumers are also called as subscribers or readers

Consumers subscribe to one or more topics and reads the messages in the order in which they were produced

Consumers keeps track of which messages it has already consumed by keeping track of the offset of messges

Apache Kafka

Outline

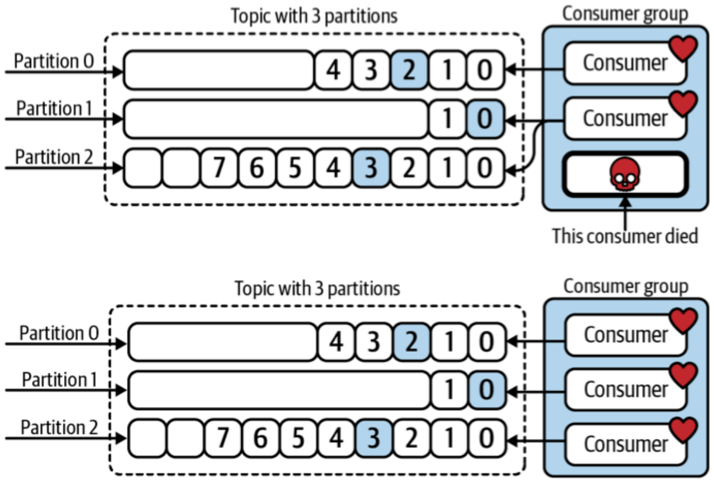
- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

Consumer Groups

Kafka optimises the high throughput and low latency on the consumer side by allowing parallel processing

Consumer groups are made up of multiple cooperating consumers

Membership of these groups can change over time, for example new consumers go online and existing consumers go offline



For each group, a special broker acts as a group coordinator (responsible for heartbeat checking and load rebalance)

Apache Kafka

Outline

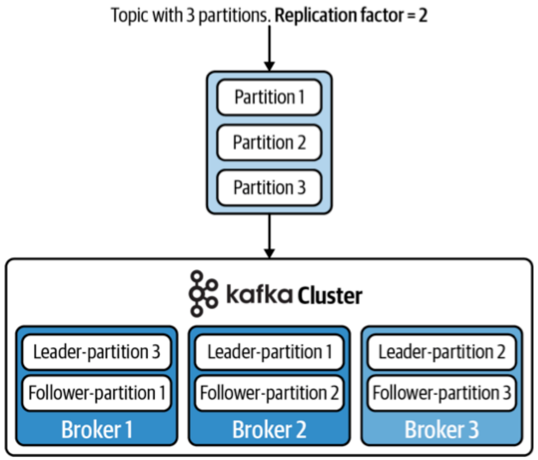
- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

Brokers and Clusters

A broker is a single Kafka server

A broker receives messages from producers, assigns offsets to them, and commits the messages to persistent storage

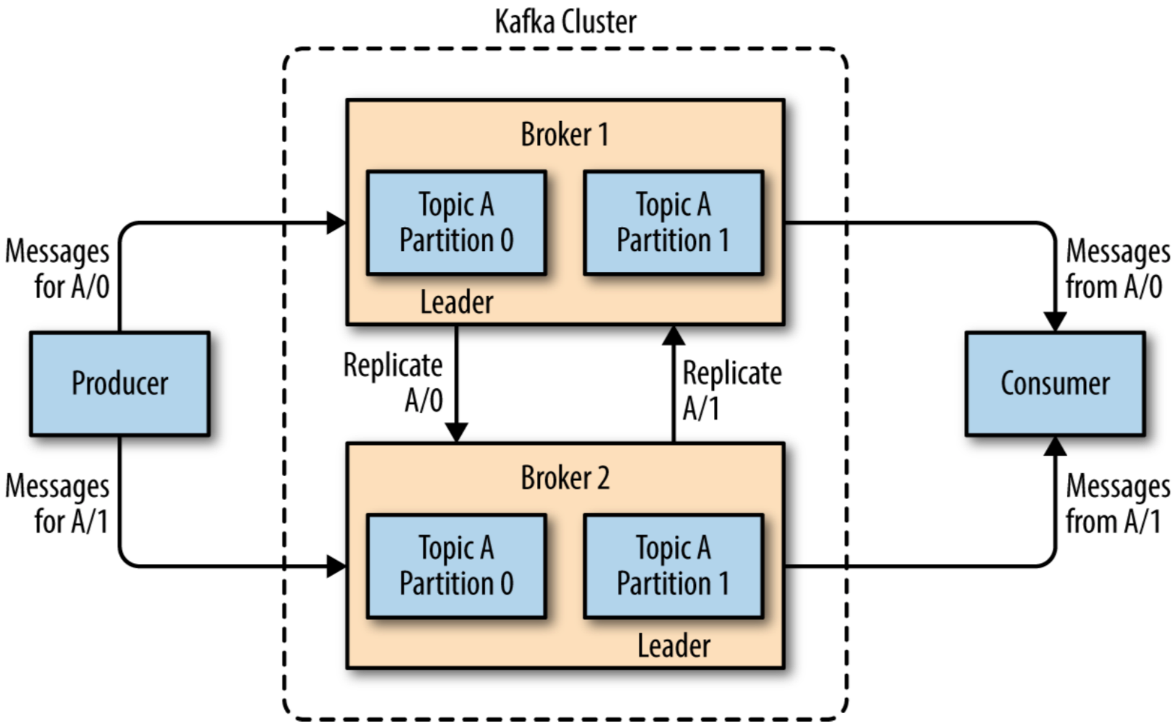
A broker services consumers, responding to fetch requests for partitions and responding with the messages that have been committed to persistent storage



Brokers are designed to operate as part of a cluster

Brokers and Clusters

Replication of partitions in a cluster



Apache Kafka

Outline

- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

Why Kafka ?

What makes Apache Kafka a good data stream processing system ?

- Multiple producers: the system is able to handle multiple producers independently they use many topics or the same topics; it makes the system ideal for aggregating data from many frontend systems
- Multiple consumers: the consumers can choose to operate as part of a group and share a stream, assuring that the entire group processes a given message only once.
- Persistent storage retention: durable message retention means that consumers do not always need to work in real time, messages are committed to persistent storage and stored with configurable retention rules
- Scalability: users can start with a single broker and later on expand to a small development cluster of three brokers, and then move into production with a larger cluster of tens or even hundreds of brokers
- High Performance: producers, consumers, and brokers can all be scaled out to handle very large message streams

Apache Kafka

Outline

- Meet Apache Kafka
- Streaming Platform
- How Streams are Stored ?
- Messages and Batches
- Schemas
- Topics and Partitions
- Events
- Producers and Consumers
- Consumer Groups
- Brokers and Clusters
- Why Kafka ?
- Integration of Spark and Kafka

Integration of Spark and Kafka

Start Apache Kafka

```
                                                              Start ZooKeeper
$CONFLUENT_HOME/bin/zookeeper-server-start
                                    $CONFLUENT_HOME/etc/kafka/zookeeper.properties
```

```
                                                           Start Kafka Broker
$CONFLUENT_HOME/bin/kafka-server-start $CONFLUENT_HOME/etc/kafka/server.properties
```

Run Kafka Producer Shell

```
                                                               Start Producer
$CONFLUENT_HOME/bin/kafka-console-producer --broker-list localhost:9092
                                                            --topic json_topic
> {"id": 1, "firstname": "James", "lastname": "Bond", "age": 35}
> {"id": 2, "firstname": "Harry", "lastname": "Potter", "age": 16}
> {"id": 3, "firstname": "Bobin", "lastname": "Hook", "age": 37}
```

Finding Topics

```
$CONFLUENT_HOME/bin/kafka-topics --list --zookeeper localhost:2181   Find Topics
```

Integration of Spark and Kafka

Load Kafka Topic to Spark Stream

Start Spark Shell

```
$SPARK_HOME/bin/spark-shell --master local
                       --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.3
```

Define a structured stream `in_ds` to which we load the Kafka topic `json_topic`

Create Structured Stream

```
val in_ds = spark.readStream
                .format("kafka")
                .option("kafka.bootstrap.servers", "localhost:9092")
                .option("subscribe", "json_topic")
                .option("startingOffsets", "earliest") // from starting
                .load()
in_ds.printSchema() // show all fields in the kafka topic
```

Integration of Spark and Kafka

Write Spark Stream to Console

Convert the binary value to String using selectExpr()

```
val str_ds = in_ds.selectExpr("CAST(value AS STRING)")
```

Extract the value to DataFrame and convert to DataFrame columns by using a custom schema

```
import org.apache.spark.sql.types._
val schema = new StructType().add("id",IntegerType)
                             .add("firstname",StringType)
                             .add("lastname",StringType)
                             .add("age",IntegerType)
val json_ds = str_ds.select(from_json(col("value"), schema)
                    .as("data"))
                    .select("data.*")
```

Compute and return the results to console

```
val age_ds = json_ds.groupBy().agg(avg("age").as("avg_age"), count("*").as("count"))
age_ds.writeStream
      .format("console")
      .outputMode("complete")
      .start()
```

Integration of Spark and Kafka

Write Spark Stream to Kafka Topic

Write the results from Spark to a new Kafka topic

```
sc.setLogLevel("ERROR")
val in_ds = spark.readStream
                 .format("kafka")
                 .option("kafka.bootstrap.servers", "localhost:9092")
                 .option("subscribe", "json_topic")
                 .option("startingOffsets", "earliest") // From starting
                 .load()
val str_ds = in_ds.selectExpr("CAST(value AS STRING)")
import org.apache.spark.sql.types._
val schema = new StructType().add("id",IntegerType)
                             .add("firstname",StringType)
                             .add("lastname",StringType)
                             .add("age",IntegerType)
val json_ds = personStringDF.select(from_json(col("value"), schema)
                            .as("data"))
                            .select("data.*")
val age_ds = json_ds.groupBy()
                    .agg(avg("age").as("avg_age"), count("*").as("count"))
                    .selectExpr("CAST(count AS STRING) AS key", "to_json(struct(*)) AS value")
age_ds.writeStream
      .format("kafka")
      .outputMode("complete")
      .option("kafka.bootstrap.servers", "localhost:9092")
      .option("topic", "avg_age")
      .option("checkpointLocation", "/tmp/checkpoint") //checkpoint dir
      .start()
```

Integration of Spark and Kafka

Run Kafka Consumer Shell

Retrieve the topic avg_age

```
$CONFLUENT_HOME/bin/kafka-console-consumer --topic avg_age --from-beginning \
--bootstrap-server localhost:9092 --property print.key=true \
--property key.deserializer=org.apache.kafka.common.serialization.StringDeserializer \
--property value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
```

References

Apache Kafka

Confluent Documentation Apache Kafka

Neha Narkhede, Gwen Shapira, and Todd Palino, Kafka: The Definitive Guide, O'Reilly 2017