LECTURER: Anna Androvitsanea

DATA ENGINEERING

Database Engineering — Unit 2

Dr.-Ing Anna Androvitsanea

IU Internationale Hochschule GmbH

February 6, 2024

TOPIC OUTLINE

Unit 2

DATA PROCESSING AT SCALE

STUDY GOALS

- Explain the differences between real-time, batch and stream processing.
- Understand application areas and fundamental concepts of batch processing.
- Use MapReduce for batch processing.
- Know application areas and fundamental concepts of stream processing.
- Using Apache Kafka and Spark Streaming for Stream Processing.

RECAP OF THE UNIT'S KEY TOPICS

Batch Processing

- Steps of a batch job
- Statuses and decisions in a batch job
- HDFS
- MapReduce

Stream Processing

- Systems techniques
- Modern stream processing systeme
- Apache Kafka
- Spark Streaming

Batch Processing: Overview - 1

- Definition: Processing large volumes of data at once, typically in a non-interactive fashion where data is collected over a period of time and processed in a single, extensive operation.
- Characteristics:
  - **High Throughput:** Optimized to process large batches of data efficiently, making it ideal for scenarios where the speed of individual transactions is less critical than the overall processing speed for large volumes of data.
  - **Resource Intensive:** Requires significant computational resources as the processing is done in large, consolidated chunks. The system is designed to maximize resource utilization during the processing window.
  - **Delayed Processing:** Unlike real-time processing systems, batch processing does not process data immediately as it arrives. Instead, data is collected over a specific period (e.g., end of day, weekly) before being processed.

Batch Processing: Overview - 2

- Use cases:
  - Financial institutions processing transactions overnight.
  - E-commerce platforms analyzing daily sales and customer behavior.
  - Healthcare systems compiling and analyzing patient data for research and reporting purposes.
- Advantage: Efficient for processing very large data sets where immediate data processing is not required, allowing organizations to utilize off-peak hours for data processing to optimize costs and resources.

Disadvantages of Batch Processing - 1

- **Latency in Data Processing:** Since data is processed in batches at predetermined intervals, there's an inherent delay in obtaining results. This can be problematic for applications requiring real-time analysis or immediate decision-making.
- **Complex Error Handling:** Errors detected during batch processing can be challenging to diagnose and correct, especially if they are found late in the batch. This can lead to time-consuming reruns and potentially affect subsequent batches.
- **Inflexibility:** Batch processing systems are often designed with specific tasks in mind and can be less adaptable to changes in data formats, processing logic, or scheduling requirements without significant reconfiguration or redevelopment.

Disadvantages of Batch Processing - 2

- **Resource Intensive During Execution:** While efficient for processing large volumes of data, batch jobs can consume substantial computational resources while they run, potentially impacting the performance of other systems or processes sharing the same infrastructure.
- **Difficulty in Processing Streaming Data:** Batch systems are not designed to handle continuous data streams effectively, making them unsuitable for scenarios where data must be processed and acted upon as it arrives in real-time.
- **Data Freshness:** Given the interval-based nature of batch processing, the data used for decision-making may not always reflect the most current state, leading to decisions made on outdated information, particularly in fast-changing environments.

Hadoop and MapReduce: Batch Processing

- **Hadoop**: A framework designed for distributed storage and batch processing of big data. Uses Hadoop Distributed File System (HDFS) for storage and supports large-scale data processing tasks across multiple computing nodes.
- **MapReduce**: A core component of Apache Hadoop, it is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster.
- **Batch Processing Focus**: Both are fundamentally designed for batch processing, handling large volumes of data in bulk, ideal for tasks like daily logs analysis and historical data processing.

Introduction to MapReduce

- Core component of the Apache Hadoop framework.
- Designed for processing large volumes of data in a distributed computing environment.
- Enables massive scalability across hundreds or thousands of servers.
- Simplifies computation across vast amounts of data in an efficient, reliable, fault-tolerant manner.

How MapReduce Works

- **Map Phase:**
  - Divides input data into independent chunks.
  - Processes chunks in parallel, producing key-value pairs.
  - Output pairs are sorted by Hadoop for the Reduce phase.
- **Reduce Phase:**
  - Takes sorted output from Map phase.
  - Aggregates data tuples based on keys.
  - Produces final output, stored in HDFS.

Key Features of MapReduce

- **Scalability:** Scales data processing over multiple computing nodes.
- **Fault Tolerance:** Handles failures at the application layer, ensuring job completion.
- **Flexibility:** Processes data in various formats from multiple sources, including HDFS, HBase, or external systems.
- **Optimization:** Manages task scheduling, data partitioning, and redundancy.

Example Use Cases

- Large-scale text processing.
- Data mining and pattern matching.
- Log analysis and fraud detection.
- Index building and search.
- Data transformations and aggregation.

MapReduce and Modern Data Technologies

- MapReduce laid the groundwork for batch processing in big data.
- Newer technologies like Apache Spark offer improvements in:
  - Speed.
  - Ease of use.
  - Real-time processing capabilities.
- Despite this, MapReduce remains a foundational technology in the Hadoop ecosystem for many big data processing tasks.

Conclusion

- MapReduce is a pivotal model for distributed data processing within the Hadoop ecosystem.
- Its principles of scalability, fault tolerance, and flexibility continue to influence the development of new data processing technologies.

Hadoop MapReduce: Word Count Example - Introduction

- The Word Count program is a fundamental example that illustrates how MapReduce works in Hadoop.
- It processes a large set of text files to count the frequency of each word.
- This example is divided into two main components: the Mapper and the Reducer.

Hadoop MapReduce: Word Count Example - Mapper

```
// Mapper Class
public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

private final static IntWritable one = new IntWritable(1);
private Text word = new Text();

public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException
  StringTokenizer itr = new StringTokenizer(value.toString(
  while (itr.hasMoreTokens()) {
    word.set(itr.nextToken().toLowerCase());
    context.write(word, one);
      }
    }
}
```

- The TokenizerMapper class tokenizes the input text and emits each word with a count of 1

Hadoop MapReduce: Word Count Example - Reducer

```
// Reducer Class
public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable> values,
                   Context context
                   ) throws IOException, InterruptedExcepti
  int sum = 0;
  for (IntWritable val : values) {
    sum += val.get();
  }
  result.set(sum);
  context.write(key, result);
    }
}
```

- The IntSumReducer class aggregates the counts for each word and emits the total count per word.

Case Study: Log Analysis with Hadoop

- Problem: Analyzing large web server logs for security insights.
- Solution: Using Hadoop MapReduce to process and analyze logs.
- Outcome: Identified security breaches and optimized web performance.

Introduction to Hadoop

- Hadoop is primarily written in **Java**.
- Java is used for core components like HDFS (Hadoop Distributed File System) and the MapReduce processing framework.
- Java offers direct access to Hadoop's APIs, allowing efficient data and resource manipulation.

Support for Other Languages

- Hadoop supports applications written in other languages via frameworks and tools.
- This facilitates interaction within Hadoop's ecosystem using various programming languages.

Apache Pig

- A platform for creating MapReduce programs.
- Uses a language called Pig Latin.
- Simplifies data manipulation operations.

Apache Hive

- Enables querying data with HiveQL, a SQL-like language.
- Queries are translated into MapReduce jobs.
- Useful for data warehousing tasks.

Streaming API

- Allows writing MapReduce jobs in languages other than Java.
- Uses standard input and output for communication.
- Supports languages like Python and Ruby.

Apache HBase

- A NoSQL database on top of HDFS.
- Can be interacted with via REST, Thrift, or Avro APIs.
- Supports a wide range of languages.

Apache Spark

- Not part of Hadoop but often used in its ecosystem.
- Provides APIs in Scala, Java, Python, and R.
- Offers more flexibility in programming language choice for big data processing.

Conclusion

- Java remains the primary language for Hadoop development.
- The ecosystem's flexibility allows for the use of various languages.
- Developers can choose the best language for their needs and preferences.

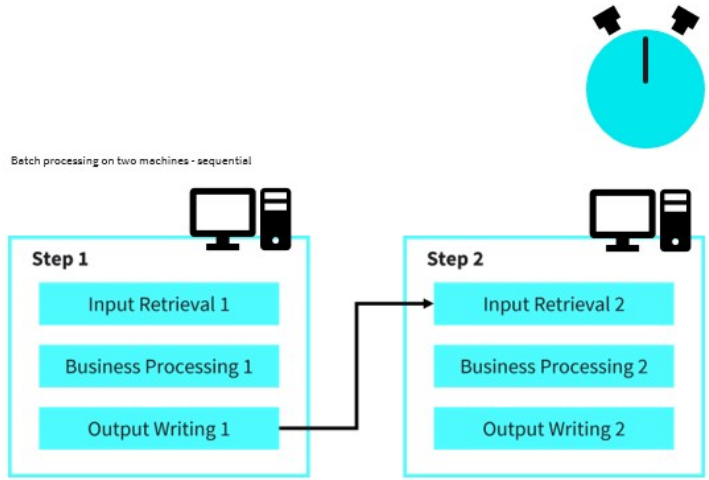
STEPS OF A BATCH JOB

- **Chunk-oriented steps**
  - break down the data into **smaller data packets (chunks)**
  - can be **parallelized** by **processing different chunks** on different computers

- **Task-oriented steps**
  - break down the processing into **several subtasks**
  - can be **parallelized** by executing **different subtasks** on different computers

Source of the text: Oracle, 2017.

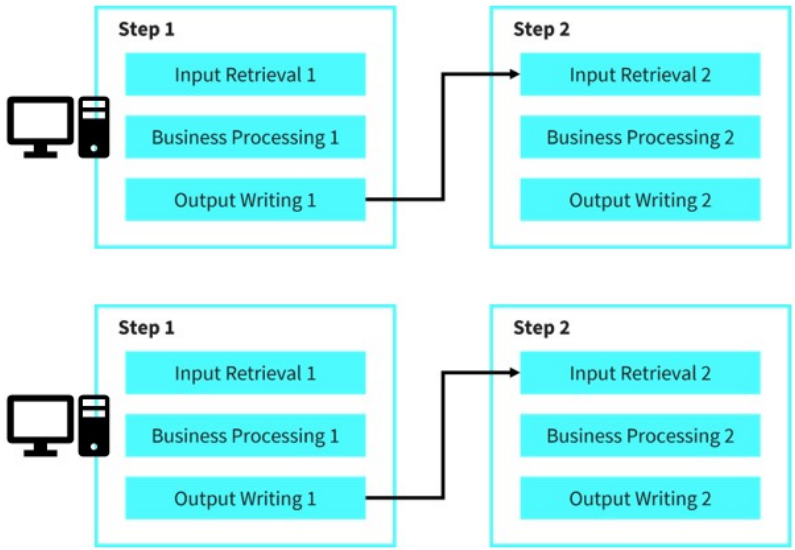
TASK-ORIENTED PARALLELISATION



Source of the image: Müller-Kett (2022), based on Oracle, 2017.

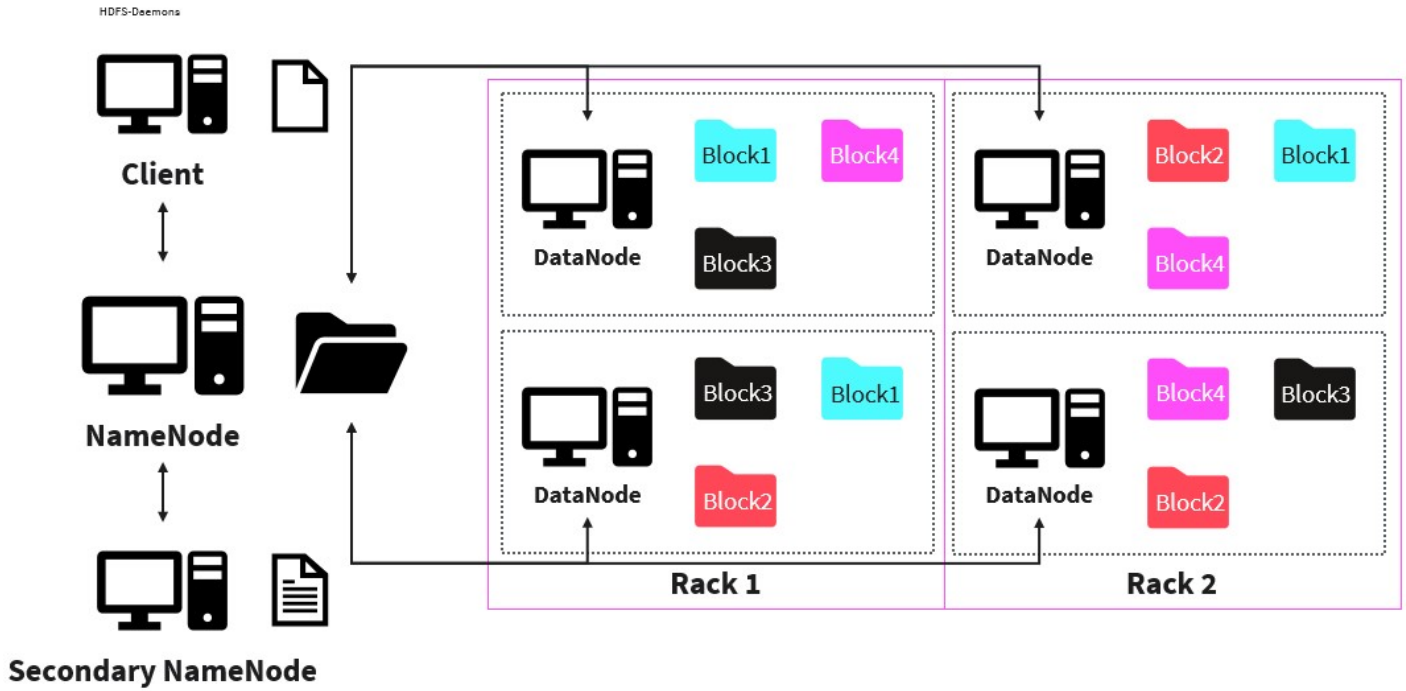
CHUNK-ORIENTED PARALLELISATION

Batch processing on two machines – parallel in the narrow sense of the word



Source of the image: Müller-Kett (2022), based on Oracle, 2017.

HDFS-ARCHITECTURE



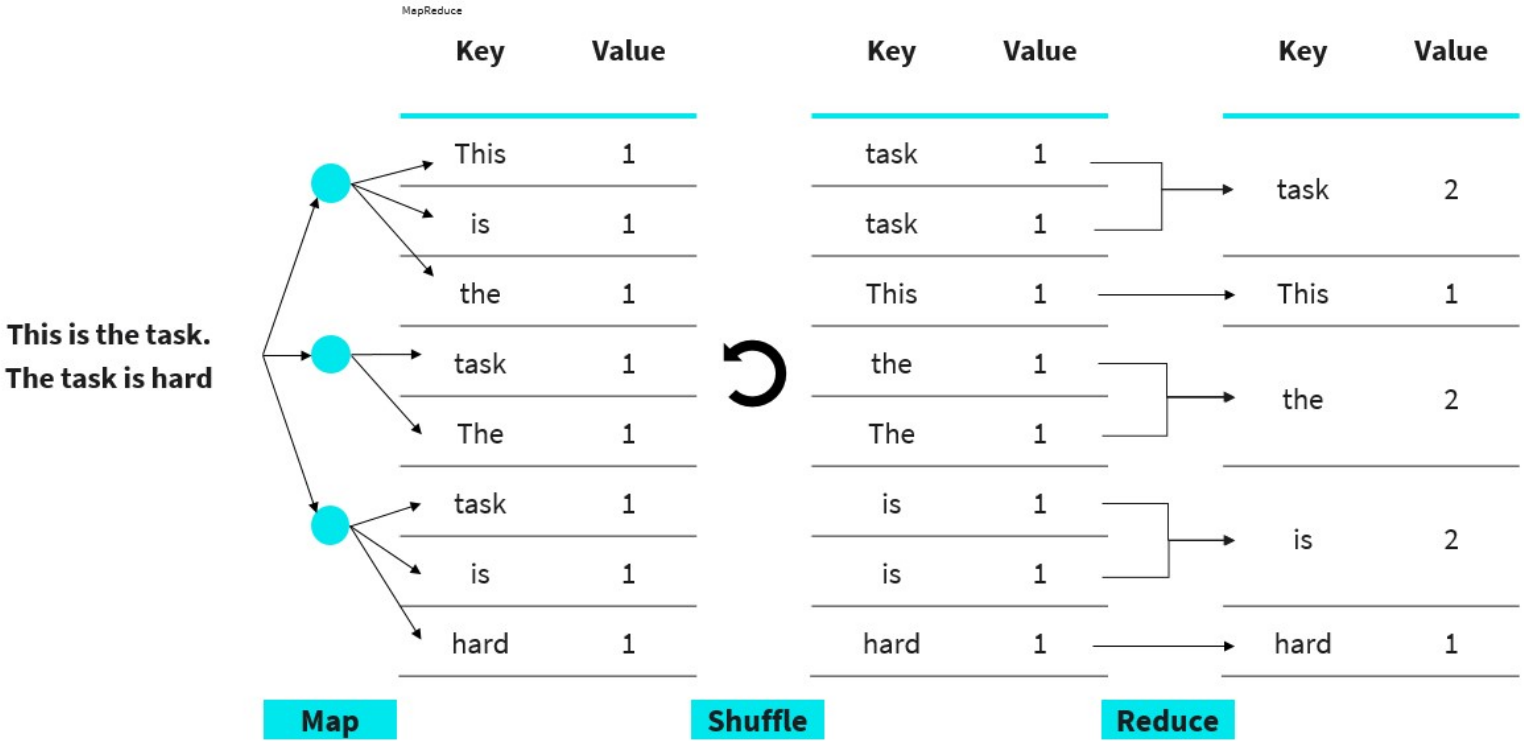
Source of the image: Müller-Kett (2022).

MAPREDUCE

- developed by **Google** to **process large amounts of data** in HDFS (Dean & Ghemawat, 2008)
- the code is brought to the **DataNodes/partitions** and executed there
- in the **mapping** part, key-value pairs are generated
- these are sorted afterwards (**shuffle**)
- in the **reduce** part, values with the same key are aggregated

Source of the text: Dean & Ghemawat, 2008.

MAPREDUCE



Source of the text: Dean & Ghemawat, 2008.
Source of the image: Müller-Kett (2022).

RECAP OF THE UNIT'S KEY TOPICS

Batch Processing

- Steps of a batch job
- Statuses and decisions in a batch job
- HDFS
- MapReduce

Stream Processing

- Systems techniques
- Modern stream processing systeme
- Apache Kafka
- Spark Streaming

Introduction to Stream Processing

- Stream Processing is the method of processing data in real time as it arrives.
- Unlike batch processing, stream processing deals with continuous data streams.
- Key for applications requiring immediate insights and actions.

Core Concepts of Stream Processing

- **Data Streams:** Continuous flow of data generated from various sources.
- **Event:** An individual record or piece of data in a stream.
- **Processing Engine:** Software that processes the data streams.

Stream Processing vs. Batch Processing

- **Real-Time vs. Delayed:** Stream processing is real-time; batch processing is delayed.
- **Data Freshness:** Stream processing ensures immediate data analysis.
- **Use Cases:** Stream processing for instant decision-making; batch for historical data analysis.

Technologies Enabling Stream Processing

- Apache Kafka for real-time data ingestion and processing.
- Apache Flink and Spark Streaming for complex event processing.
- Managed cloud services like Amazon Kinesis for scalable stream processing.

Example Use Case: Real-Time Analytics

- Real-time monitoring of financial transactions to detect fraud.
- Live dashboards for monitoring system performance.
- Instant recommendations for e-commerce based on user actions.

Challenges in Stream Processing

- Ensuring data accuracy and completeness in real-time.
- Managing large volumes of data with minimal latency.
- Handling out-of-order data and ensuring event consistency.

Conclusion

- Stream Processing plays a crucial role in enabling real-time data analysis.
- It complements batch processing by providing immediate insights and actions.
- As data volume and velocity continue to grow, stream processing becomes increasingly important for modern data architectures.

Apache Kafka Stream Processing

```
# Kafka Streams application for word count
builder = StreamsBuilder()
textLines = builder.stream("input_topic")
wordCounts = textLines.flatMapValues(lambda line: line.lowe
    .groupBy(lambda word: word)
    .count()
wordCounts.toStream().to("output_topic")
```

- Consumes text lines from ‘input_topic‘.
- Processes stream to count occurrences of each word.
- Outputs word counts to ‘output_topic‘.

Case Study: Real-time Monitoring System

- Problem: Monitoring and alerting for infrastructure performance.
- Solution: Implemented a stream processing pipeline with Apache Kafka.
- Outcome: Immediate detection and response to performance anomalies.

Introduction to Apache Kafka

- Apache Kafka is a distributed streaming platform.
- Primarily written in **Scala and Java**.
- Leverages Scala's concise syntax and functional programming along with Java's reliability.

Kafka Development Languages

- **Core Components:** Developed using Scala and Java for efficiency and scalability.
- **Client APIs:** Java is predominantly used for Kafka's client APIs, enabling production and consumption of messages.

Support for Other Languages

- Despite the Scala and Java core, Kafka supports clients in various programming languages:
  - Python (confluent-kafka-python, kafka-python)
  - C/C++ (librdkafka)
  - Go, and others
- Facilitates Kafka's integration into diverse development environments.

Stream Processing with Kafka

- **Kafka Streams:** A stream processing library included with Kafka.
- Written in Java, providing a high-level API for building applications.

Kafka Connect

- For building data import/export pipelines.
- Also Java-based, supporting a wide range of connectors for different data sources and sinks.

Conclusion

- Apache Kafka's core is Scala and Java, ensuring robust and scalable streaming capabilities.
- The ecosystem supports multiple languages, making Kafka accessible to a wide developer audience.
- Kafka's flexibility and language support cater to diverse software architectures and developer preferences.

PREDECESSOR OF MODERN STREAM PROCESSING SYSTEMS

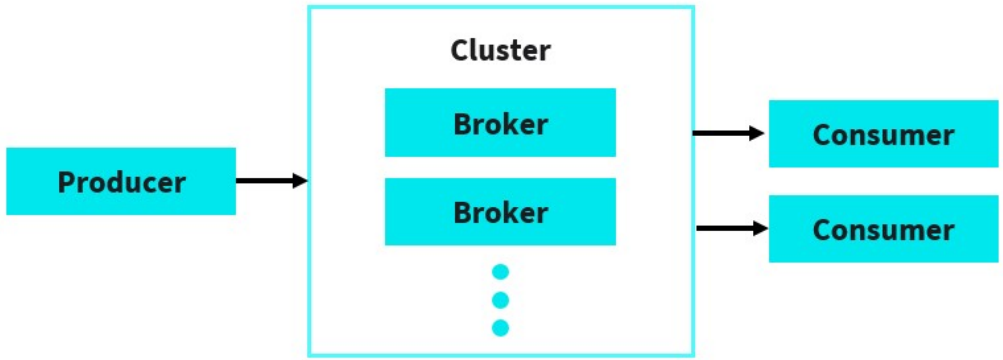
- **Active databases**
  - Event-condition-action
  - closed/open active databases
- **Continuous query systems**
  - Continuous Query (CQ)
  - Query-trigger-stop
- **Publish-subscribe systems**
  - Publisher-Broker-Consumer
  - topic-based / content-based systems
- **Complex event processing systems**
  - Combinations of simple events

- **Predecessors** of modern systems are **limited** in the scope of data processing
- **Stream processing paradigm**
  - Streaming data tuples
  - Operators
  - Stream connections
  - Stream processing applications (SPA)
- **Stream Processing System (SPS)**
  - Development environment
  - Runtime environment

APACHE KAFKA

- originally developed as a **message queue** system by **LinkedIn**
- **immutable distributed commit log**

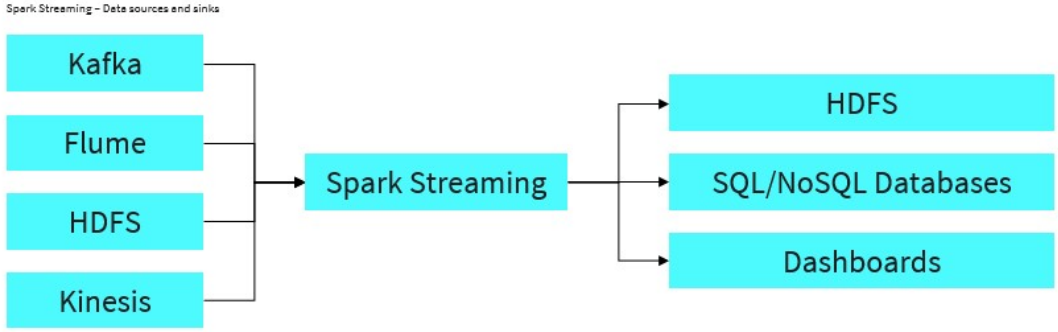
Apache Kafka High-Level concept



Source of the image: Müller-Kett (2022).

Extension library to **Apache Spark**

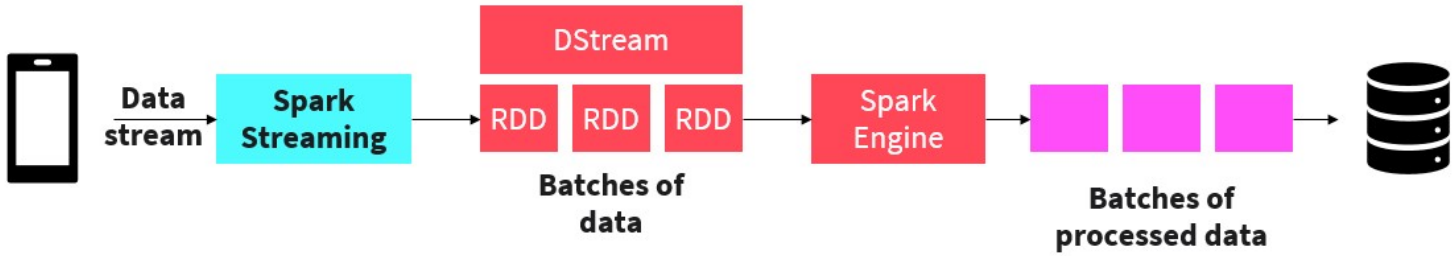
- **fault-tolerant** stream processing
- **scalable**, **high throughput**
- **integrates** seamlessly with **Apache Spark** and the corresponding libraries
- API supports **numerous data sources and sinks**



Source of the image: Müller-Kett (2022).

SPARK STREAMING WORKFLOW

Spark Streaming Workflow



RDD = Resilient Distributed Dataset
DStream = Discretized Stream

Source of the image: Müller-Kett (2022).

LATENCY

- **Spark Streaming** ~ milliseconds to a few seconds
- **Kafka** ~ below milliseconds

Reason to choose **Spark Streaming over Kafka**

- Seamless **integration** into the **Spark** environment
- **Rich APIs** which allow to connect to a plethora of data sources and sinks
- ...

Conclusion

- Batch and stream processing serve different but complementary roles.
- Choice of technology depends on the specific requirements for latency, throughput, and processing semantics.
- Both technologies are essential for building scalable, data-driven applications.

SESSION 2

TRANSFER TASK

TRANSFER TASK

A start-up that sells **sustainable products in smaller stores** has been very successful in recent years. As a result, more stores are to be opened worldwide. As a Data Engineer, you will be tasked with **designing the data system that stores and processes data** about the products offered and their suppliers.

Every morning, store managers will have access to a **report** showing **which goods** are available **in which quantities** and which goods need to be ordered from **which suppliers**. In addition, in order to **identify trends** at an early stage, there should be a warning system that monitors **every minute** whether there is particularly **high demand** for an article worldwide. This warning system should be **compatible with as many external services as possible**, since it is to be integrated into a larger system landscape.

As a team, create a **conceptual plan** for the data system that meets these requirements. In doing so, address the **storage and processing** of the data and **make concrete proposals for the technology** to be used. **Describe briefly** and in **simple terms** how these work in the example of this concrete use case.

TRANSFER TASK
PRESENTATION OF THE RESULTS

Please present your results.
The results will be discussed in plenary.

TRANSFER TASK – SAMPLE SOLUTION

„ **Every morning**, store managers will have access to a **report** showing **which goods** are available **in which quantities** and which goods need to be ordered from **which suppliers**.“

- Since a report is to be available every morning, **batch processing** lends itself to meeting this requirement for the system
- For example, the **Hadoop Distributed File System (HDFS)** could be used to store the data
- The data (inventory and suppliers) is stored on different **DataNodes within a cluster**
- **Parallel processing** of the data enables short processing times for large data volumes
- High system **reliability** can be achieved by **replicating** the data on different DataNodes
- As an abstraction layer, we connect to the NameNode, which makes it easier to work with the file system
- As a result, we see only one "ordinary" file directory where we can store the data

TRANSFER TASK – SAMPLE SOLUTION

- For **batch processing** of the data, **MapReduce** could be used, for example
  - In the first part, the **mapping** part, of a simple MapReduce job, **key-value pairs** could be generated for this use case, where the **individual goods** represent the key and the **respective available quantity** the value
  - In the **Reduce** part, this information is then **summed up for each article**
  - A **second MapReduce job** could be used, for example, to calculate the **quantity of goods per store** or the **quantity to be ordered per supplier**

TRANSFER TASK – SAMPLE SOLUTION

„ In addition, in order to **identify trends** at an early stage, there should be a warning system that monitors **every minute** whether there is particularly **high demand** for an article worldwide. This warning system should be **compatible with as many external services as possible**, since it is to be integrated into a larger system landscape.“

- **Stream processing** is the ideal solution for this application
- Since the processing is to take place **every minute**, very short **latency** times play a rather **subordinate role**
- Since the system should also be **compatible** with as **many external services** as possible, **Spark Streaming** is a suitable platform for this system requirement

TRANSFER TASK – SAMPLE SOLUTION

- Here, the **cash registers** of the individual stores could be considered as **data sources** and the **external services** (e.g., an email client that sends alerts and/or a database in which the results are persisted) as **data sinks**

- Within **Spark Streaming**, the **calculations** are performed

- For example, an **anomaly detection** mechanism could be used to detect unusually high demand in an automated manner

- These computations are performed on **Dstreams** (Discretized Streams), which provide an abstraction of the underlying **RDDs** (Resilient Distributed Datasets)

LEARNING CONTROL QUESTIONS

1. Which daemon in HDFS is the master node?
   a) DataNode
   b) JobTracker
   c) NameNode
   d) NodeTracker

2. What is the name of a category or feed name in Kafka to which the news are published?
   a) Topic
   b) Tupel
   c) Consumer
   d) Producer

3. What defines the structure of data tuples?
   a) Schema
   b) Tabel
   c) Data tupel
   d) Data stream

LIST OF SOURCES

Text:

Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM, 51*(1). 107—113. https://doi.org/10.1145/1327452.1327492

Oracle. (2017). *Introduction to batch processing*. https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/BatchProcessing/BatchProcessing.html

Image:

Müller-Kett (2022).

Müller-Kett (2022), based on Oracle, 2017.