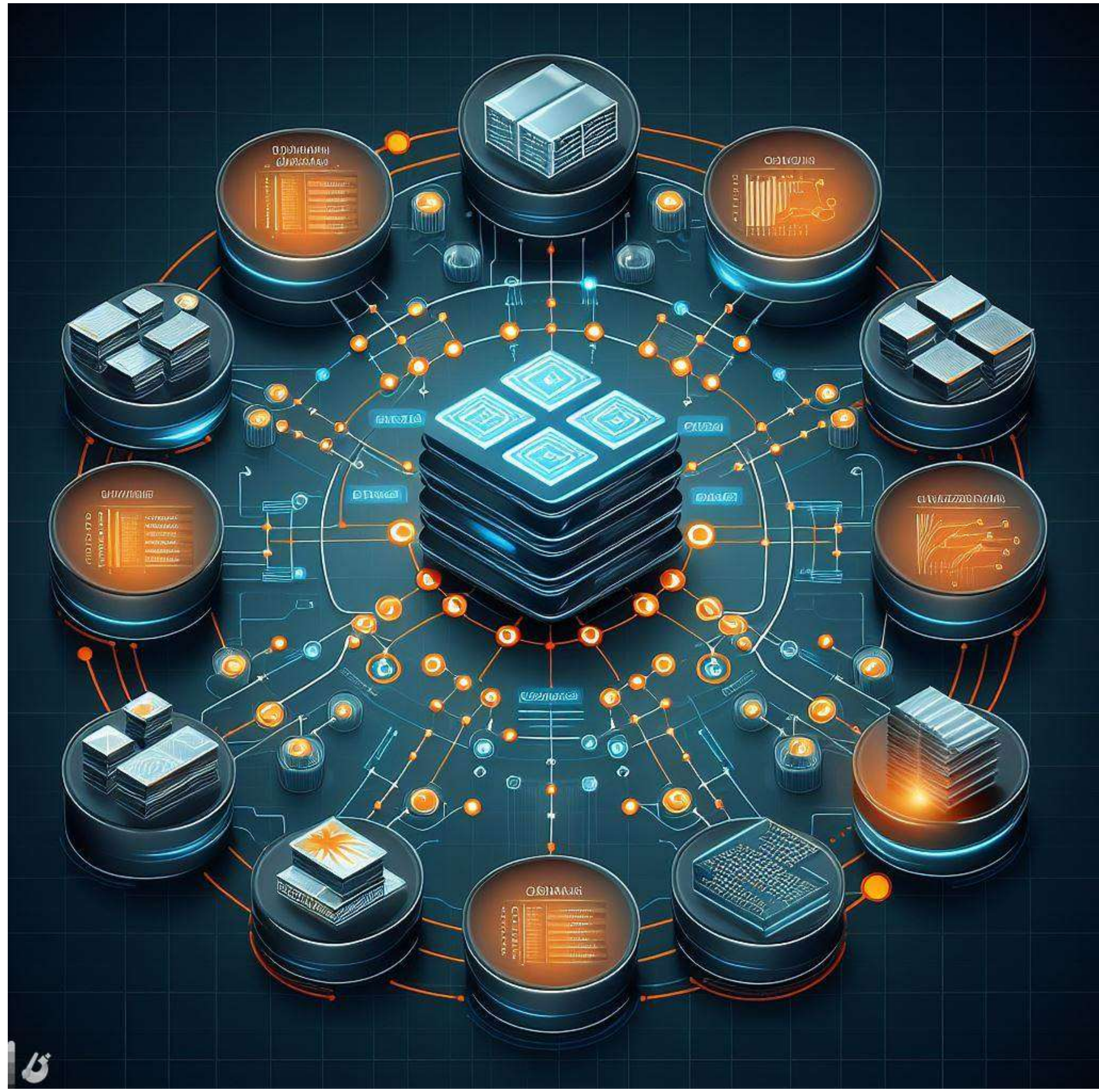
Parallel Processing with Map Reduce



Chapter 07

Instructor: Houssein Dhayne
houssein.dhayne@ul.edu.lb

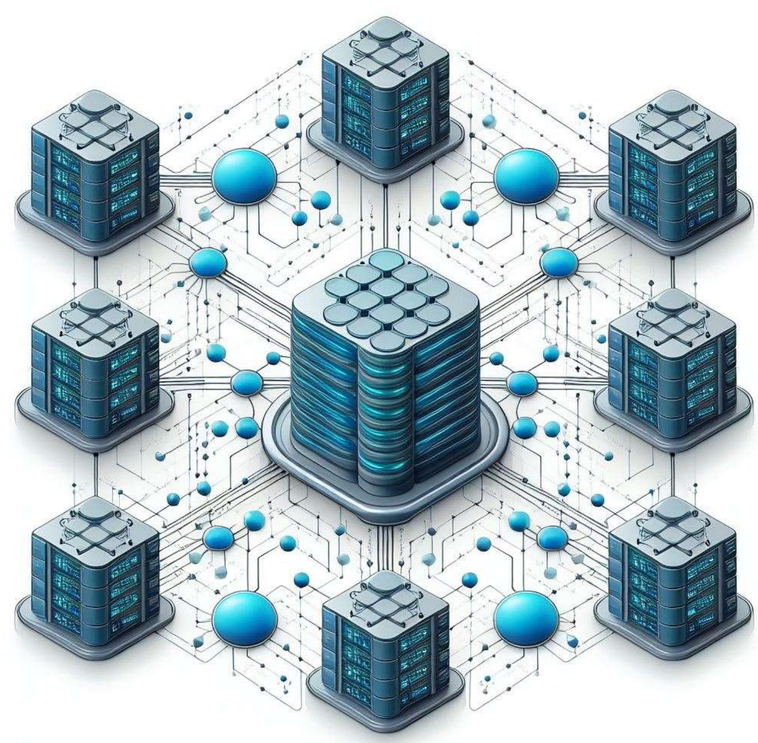
Table of Contents

Introduction to Parallel Processing

- Parallel processing is a powerful approach to efficiently handle large amounts of data.
- The ability to distribute tasks across multiple computing devices allows for faster data processing.

Need for Parallel Processing in Big Data

- The increasing volume of big data requires innovative solutions for quick and efficient processing.
- Parallel processing enables the simultaneous execution of tasks across multiple computing devices.
- This approach is essential for handling vast datasets within reasonable time frames, addressing the challenges posed by big data.
- In this context, we explore the significance of parallel processing in managing big data effectively.



How Google Search Uses MapReduce

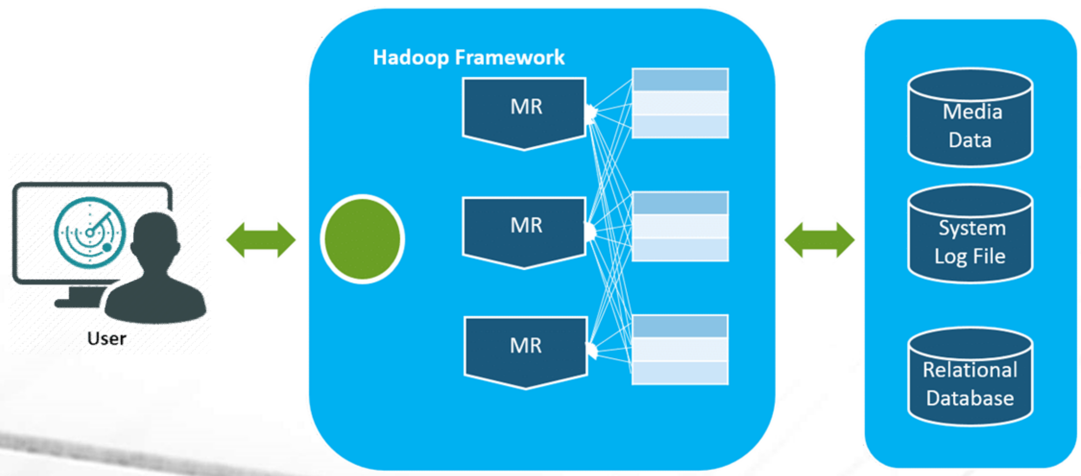
- Google Search employs the MapReduce algorithm for efficient and rapid search query processing.
- Search queries are broken into sub-queries using keywords, and a Map program is generated for each sub-query.
- These Map programs are distributed to data nodes globally, where they process data and return results, demonstrating the scalability and effectiveness of MapReduce in real-world applications.

Overview of MapReduce

- MapReduce serves as a parallel programming framework for large-scale data processing.
- It operates with minimal data movement on distributed file systems like Hadoop clusters, enabling near real-time results.
- The prerequisite for MapReduce programming includes applications suitable for parallel processing and data expressible in key-value pairs.

Prerequisites for MapReduce Programming

- Two major prerequisites for MapReduce programming are:
  - The application must lend itself to parallel programming.
  - The data for the application must be expressible in key-value pairs.
- Drawing parallels with UNIX sequences, the structure of MapReduce processing is introduced, emphasizing the division of tasks and the intermediate results.



Key-Pair Data Structure in MapReduce

- MapReduce relies on a key-pair data structure, where data is represented as key-value pairs.
- This structure facilitates the distribution of tasks and results among computing devices.
- Understanding and leveraging the key-pair data structure is crucial for effective implementation of MapReduce programs.

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

MapReduce Processing Structure - UNIX Sequence Comparison

- MapReduce processing can be likened to UNIX sequences (pipes) structure.
- A concrete example is illustrated with a UNIX command:
  - grep | sort | count myfile.txt
- This comparison aids in understanding the flow of MapReduce tasks, highlighting the Map, Sort, and Reduce steps.

Grep
We
are
going
to
a
picnic
near
our
house
Many
of
our
friends
are
coming
You

Sort
a
Are
Are
Are
Coming
Friends
Fun
Going
Have
House
Join
Many
Near
Of
Our
Our

WordCount	
A	1
Are	3
Coming	1
Friends	1
Fun	1
Going	1
Have	1
House	1
Join	1
Many	1
Near	1
Of	1
Our	2
Picnic	1
To	2
Us	1

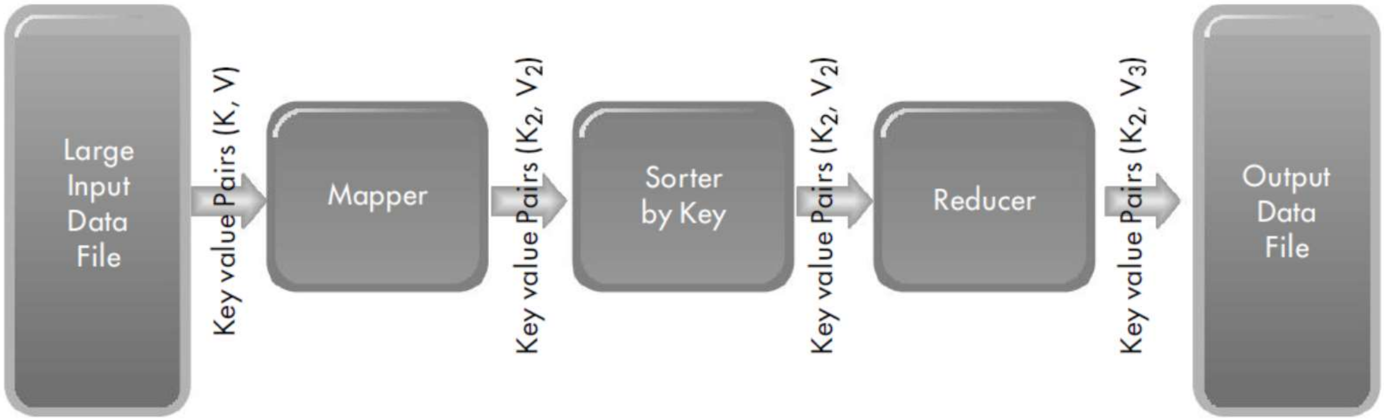
Program Structure for MapReduce Programs

- MapReduce programs are structured with a Map program and a Reduce program.
- The Map program processes input data and produces intermediate key-value pairs.
- The Reduce program takes these intermediate results, performs further processing, and produces the final output.
- This structured approach ensures efficient parallel processing of large datasets.

Key Characteristics of MapReduce Data Processing

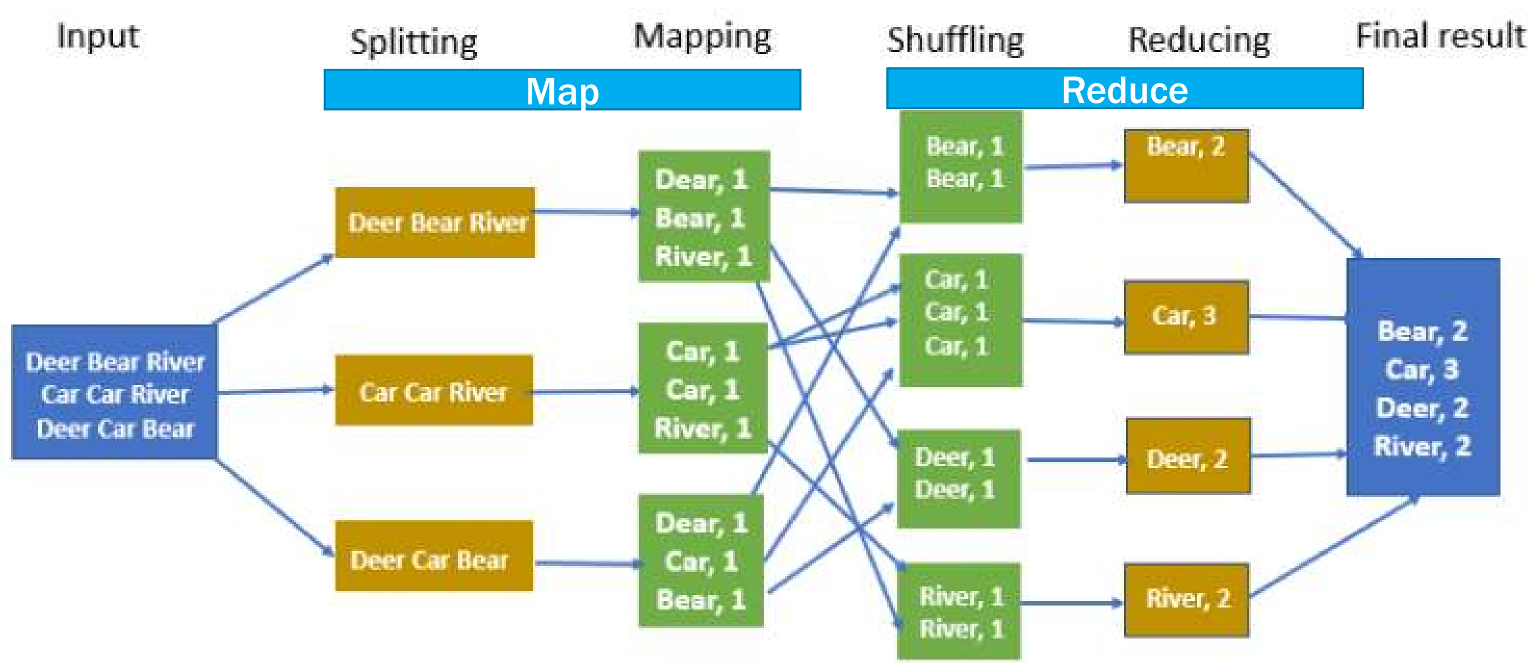
Key characteristics of MapReduce data processing are:

- Map step distributes the full job into smaller tasks processed on separate computers.
- Intermediate results from the Map step are considered as key2-value2 pairs.
- The Reduce step reads and combines the intermediate results to produce the final output in key2-value3 format.



Sample MapReduce Application - WordCount

- A practical example of a MapReduce application is presented: WordCount.
- The scenario involves identifying unique words and their frequency in a text.



We are going to a picnic near our house.	1

Many of our friends are coming.	1

You are welcome to join us.	1

We will have fun.	1

Key2	**Value2**
we	1
are	1
going	1
to	1
a	1
picnic	1
near	1
our	1
house	1

Key2	**Value2**
many	1
Of	1
Our	1
friends	1
Are	1
coming	1

Key2	**Value2**
you	1
Are	1
Welcome	1
To	1
Join	1
Us	1

Key2	**Value2**
we	1
will	1
have	1
fun	1

- The sort process inherent within MapReduce will sort each of the intermediate files, and produce the following sorted key2-value2 pairs:

Key2	Value2
a	1
Are	1
Going	1
House	1
Near	1
Our	1
Picnic	1
to	1
We	1

Key2	Value2
Are	1
coming	1
friends	1
many	1
Of	1
our	1

Key2	Value2
Are	1
Join	1
To	1
Us	1
welcome	1
you	1

Key2	Value2
fun	1
have	1
we	1
will	1

Reduce

- The Reduce function will read the sorted intermediate files, and combine the counts for all the unique words, to produce the following output.
- The keys remain the same as in the intermediate results.
- However, the values change as counts from each of the intermediate files are added up for each key.
- For example, the count for the word ‘are’ goes up to 3.

Key2	Value3
a	1
are	3
coming	1
friends	1
fun	1
going	1
have	1
house	1
join	1
many	1
near	1
of	1
our	2
picnic	1
to	2
us	1

MAPREDUCE PROGRAMMING

MapReduce Data Types and Formats

- MapReduce operates with a simple data processing model using key-value pairs.

$$\text{map: } (K1, V1) \rightarrow \text{list } (K2, V2)$$

$$\text{reduce: } (K2, \text{list}(V2)) \rightarrow \text{list } (K2, V3)$$

- Map and reduce functions have specific input and output types.
- Data formats, such as flat text files and databases, can be processed by MapReduce.

Writing MapReduce Programs -1-

- In Java, the Map function is represented by the generic Mapper class.
- It uses four parameters: (input key, input value, output key, output value).
- This class uses an abstract map ( ) method. This method receives the input key and input value.
- It would normally produce output key and output value.
- A Mapper commonly performs input format parsing, projection (selecting the relevant fields), and filtering (selecting the records of interest).
- The Reducer typically combines (adds or averages) those values.

Writing MapReduce Programs -2-

```
// Mapper: Emits (word, 1) for each occurrence
public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  public void map(...) {
    // Split input line into words and emit (word,1)
  }
}

// Reducer: Sums counts for each word
public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(...) {
    int sum = 0;
    for (IntWritable value : values) { sum += value.get(); }
    context.write(key, new IntWritable(sum));
  }
}
```

Writing MapReduce Programs - step-by-step logic

- The big document is split into many segments. The Map step is run on each segment of data. The output will be a set of (key, value) pairs. In this case, the key will be a word in the document.
- The system will gather the (key, value) pair outputs from all the mappers, and will sort them by key. The sorted list itself may then be split into a few segments.
- A Reducer task will read the sorted list and produce a combined list of word counts.

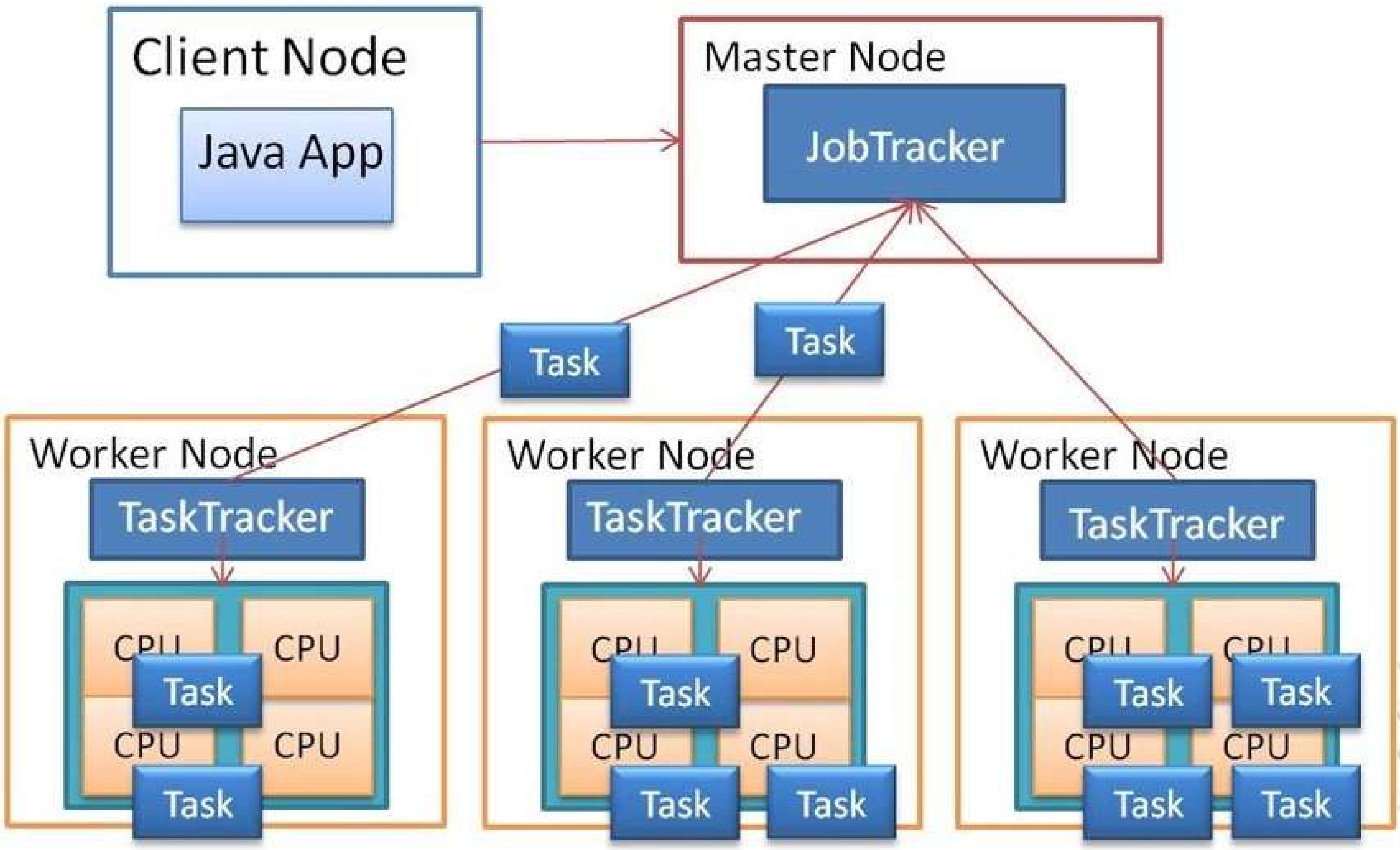
```
map(String key, String value):
   for each word w in value:
   EmitIntermediate(w, "1");
reduce(String key, Iterator values):
   int result = 0;
   for each v in values:
      result += ParseInt(v);
Emit(AsString(result));
```

MapReduce Program Optimization - Using Combiners

- To optimize MapReduce programs, using combiners is introduced.
- Combiners are functions that operate on the output of the Map function before it reaches the Reducer.
- For example, a combiner is introduced with the combine method. The combiner performs a partial aggregation of counts for each word locally on each Mapper node before sending the data to the Reducer.
  - This helps reduce the amount of data transferred to the Reducer, improving overall efficiency. The combine method has the same logic as the reduce method in the Reducer, but it operates locally on each Mapper node.

MAPREDUCE JOBS EXECUTION

Synchronization of Tasks in MapReduce



MapReduce Job Execution– Overview

- **Job Components:**
  - MapReduce job comprises the Map and Reduce programs along with input datasets.

- **Job Coordination:**
  - A master program (Job Tracker on the NameNode) continuously monitors job progress.
  - Job coordination ensures that the distributed tasks are executed efficiently, reliably, and in a balanced manner throughout the cluster.

- **Distributed Processing:**
  - Map and Reduce tasks are executed on DataNodes hosting data fragments.

- **Resource Management:**
  - Communication and resource allocation among nodes are managed by YARN.

Job Tracker, YARN, and Data Locality

- **Job Tracker:**
  - Resides on the NameNode; responsible for coordinating job execution.
  - Assigns tasks based on the location of data to minimize network traffic.
- **YARN:**
  - Acts as the resource manager; allocates containers and schedules tasks across the cluster.
- **Data Locality:**
  - The system schedules tasks on nodes where the data already resides (e.g., if Node A holds data (x, y, z), Map tasks for that data run on Node A).

Task Execution: Map, Shuffle, and Reduce

- **Input Splits:**
  - The dataset is divided into fixed-size pieces called input splits.
  - One map task is created per input split.
- **Map Phase:**
  - Each map task applies the user-defined Map function to process records and produce intermediate key-value pairs.
- **Shuffle and Sort:**
  - Intermediate data is buffered, sorted by key, and partitioned for the reduce phase.
  - This sorting is necessary for efficient processing during the Reduce phase.
  - An optional combiner can perform local aggregation to reduce data transfer.
- **Reduce Phase:**
  - Reduce tasks merge sorted outputs from map tasks to produce the final results.

Job and Task Management

- **Task Tracker:**
  - Each DataNode runs a Task Tracker to monitor assigned Map and Reduce tasks.
  - On task completion, the Task Tracker notifies the Job Tracker.
- **Job Scheduling:**
  - YARN's scheduler dynamically allocates resources for each task based on current cluster capacity.

Managing Failures and Ensuring Resilience

- **Task Failures:**
  - If a task fails (e.g., due to a runtime error), the Task Tracker reports the error to the Job Tracker, which then reschedules the task on a different node.
- **Application Master Failures:**
  - The entire MapReduce job may be restarted if the Application Master (running on YARN) fails, up to a configurable limit.
- **Node Failures:**
  - YARN detects DataNode failures via missing heartbeats and reallocates tasks from the failed nodes to healthy ones.

These mechanisms ensure continuous processing even in the presence of hardware or software failures.

Testing MapReduce Jar: Word Count Example

Input File: wordscount.txt
Output Directory: wordscountout
Jar File: WordCount.jar
Main Class: org.WordCount.WordCountDriver

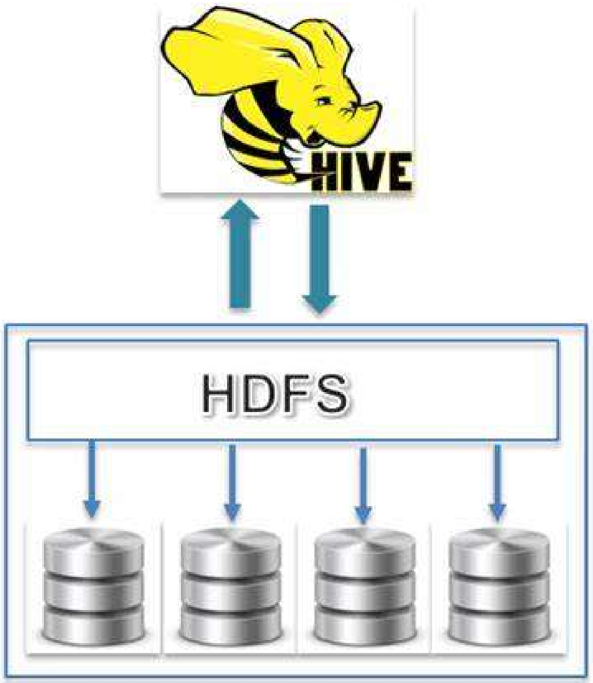
- Step 1: Upload the input file (wordscount.txt) to the Hadoop Distributed File System (HDFS).
- hdfs dfs -put "/usr/mybigdata/wordscount.txt" /user/test/
- Execute the MapReduce application by specifying the input file (wordscount.txt) and the output directory (wordscountout).
- hadoop jar WordCount.jar org.WordCount.WordCountDriver /user/test/wordscount.txt /user/test/wordscountout

Hive Overview

Data warehouse infrastructure tool to process structured data in Hadoop.

Introduction to Apache Hive

- Apache Hive is a data warehouse and ETL (Extract, Transform, Load) tool.
- Provides an SQL-like interface for users to interact with Hadoop Distributed File System (HDFS).
- Built on top of the Hadoop ecosystem, facilitating data query and analysis.
- **Apache Hive** makes querying and managing large datasets easier through SQL-like syntax called HiveQL. Hive translates these queries into **MapReduce jobs** under the hood to process data.



Development and Contributors

- Initially developed by Facebook and later adopted by Amazon, Netflix, and others.
- Delivers standard SQL functionality for analytics.
- Not suitable for Online Transactional Processing (OLTP) workloads.
- Commonly used for data warehousing, and analyzing large datasets.
- Facilitates reading, writing, and handling wide datasets stored in distributed storage.

Modes of Hive

- **Local Mode:**
  - Used in pseudo mode with a single data node.
  - Suitable for smaller datasets on a local machine.
- **MapReduce Mode:**
  - Used with multiple data nodes in a Hadoop cluster.
  - Executes queries in parallel for enhanced performance on large datasets.

Characteristics of Hive

- Databases and tables must be created before loading data.
- Hive as data warehouse is built to manage and query only structured data which is residing under tables.
- Uses directory structures to improve performance on certain queries.
- Compatible with various file formats (TEXTFILE, SEQUENCEFILE, ORC, RCFILE).

Features of Hive

- It provides indexes, including bitmap indexes to accelerate the queries.
- Built in user-defined functions (UDFs) to manipulation of strings, dates, and other data-mining tools.
- It stores schemas in a database and processes the data into the Hadoop File Distributed File System (HDFS).
  - Schema in Database: Hive stores the metadata (like table names, column types, partitions, and locations) in a Metastore — usually backed by a database like MySQL or PostgreSQL.
  - Data in HDFS: The actual data is stored in HDFS (Hadoop Distributed File System), while Hive queries act as a layer on top to process that data.

Advantages of Apache Hive

- **Scalability:**
  - Designed to handle large volumes of data.
- **SQL-Like Interface:**
  - HiveQL is similar to SQL, making it familiar for SQL users.
- **Integration with Hadoop Ecosystem:**
  - Seamless integration with other Hadoop tools like Pig, MapReduce, and Spark.
- **Supports Partitioning and Bucketing:**
  - Improves query performance by limiting data scanned.

Disadvantages of Apache Hive

- **Limited Real-time Processing:**
  - Designed for batch processing, not real-time data processing.
- **Slow Performance:**
  - Can be slower compared to traditional relational databases.
- **Steep Learning Curve:**
  - Requires knowledge of Hadoop and distributed computing.
- **Lack of Transaction Support:**
  - Does not support transactions, impacting data consistency.
- **Limited Flexibility:**
  - Specific to Hadoop, limiting usability in other environments.

Hive Language Capabilities

- Hive's SQL provides standard SQL operations such as SELECT, FROM, WHERE, JOIN, GROUP BY, and ORDER BY.
- Results can be stored in tables or HDFS files, offering flexibility.
- Comparisons between Hive and traditional relational databases highlight its advantages for specific use cases.

Creating a Database and Table

- **Create a database named 'school'**

```
CREATE DATABASE IF NOT EXISTS school;
```

- **Switch to the 'school' database**

```
USE school;
```

- **Create a table named 'student' within the 'school' database**

```
CREATE TABLE student (
  student_id INT,
  name STRING,
  age INT,
  grade STRING
);
```

Inserting Data into a Table

- Insert data into the 'student' table

```
INSERT INTO student VALUES
  (1, 'John Doe', 20, 'A'),
  (2, 'Jane Smith', 22, 'B'),
  (3, 'Bob Johnson', 21, 'C');
```

- Overwrite existing data in the 'student' table

```
INSERT OVERWRITE TABLE student VALUES
  (1, 'Alice Johnson', 23, 'A'),
  (4, 'Charlie Brown', 24, 'B'),
  (5, 'Eva White', 22, 'A');
```

INSERT OVERWRITE with Immutable Files and Blocks in Hadoop

How INSERT OVERWRITE Works with Immutable Files and Blocks in Hadoop?

- INSERT OVERWRITE involves overwriting existing data in Hive.
- Hadoop follows an immutable data model - once written, data is not modified in-place.
- New set of data files is created, and the old ones are not directly modified.

Loading Data into Hive

The following Hive script creates a student table to store structured data in a text file format. The table uses a comma (',') as a delimiter to separate fields, making it easy to load and query data efficiently.

```
Drop table student;
CREATE TABLE student (
  student_id INT,
  name STRING,
  age INT,
  grade STRING )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Loading Data into Hive

- To load a file into Hive, you typically use the LOAD DATA statement.
- This statement is used to load data from a local file or HDFS (Hadoop Distributed File System) into a Hive table.

```
-- Load data into the Student' table from a local file

LOAD DATA LOCAL INPATH '/path/to/data.txt' INTO TABLE Student;

-- Load data into the Student table from HDFS

LOAD DATA INPATH '/hdfs/path/to/data.txt' INTO TABLE student;
```

Creating and Loading Data into the wordcount Table in Hive

- CREATE TABLE wordcount (
- word STRING,
- count INT
-)
- ROW FORMAT DELIMITED
- FIELDS TERMINATED BY ','
- STORED AS TEXTFILE;

- LOAD DATA INPATH '/user/test/wordscountout/part.csv' INTO TABLE wordcount;

Conclusion

- MapReduce stands out as the leading parallel processing framework for Big Data applications.
- Its effectiveness lies in its ability to process large, divisible datasets, represented in <key, value> pair format.
- High-level languages like Hive and Pig enhance the ease of MapReduce programming.
- The presentation has covered fundamental MapReduce concepts, testing strategies, optimization techniques, and complementary tools like Hadoop Streaming and Hive.