Wednesday, 10th May 2023

DSAI, IIIT Dharwad

INSTRUCTOR

Dr.Animesh Chaturvedi

GROUP

Chava Srinivas Sai - 21BDS012
R Dhivya Prasanna - 21BDS052
Sadikh Shaik - 21BDS059
Zuhair Hasan - 21BDS060

Accelerating Big Data Analytics with Apache Spark and HDFS Cluster Scaling

Table of Contents

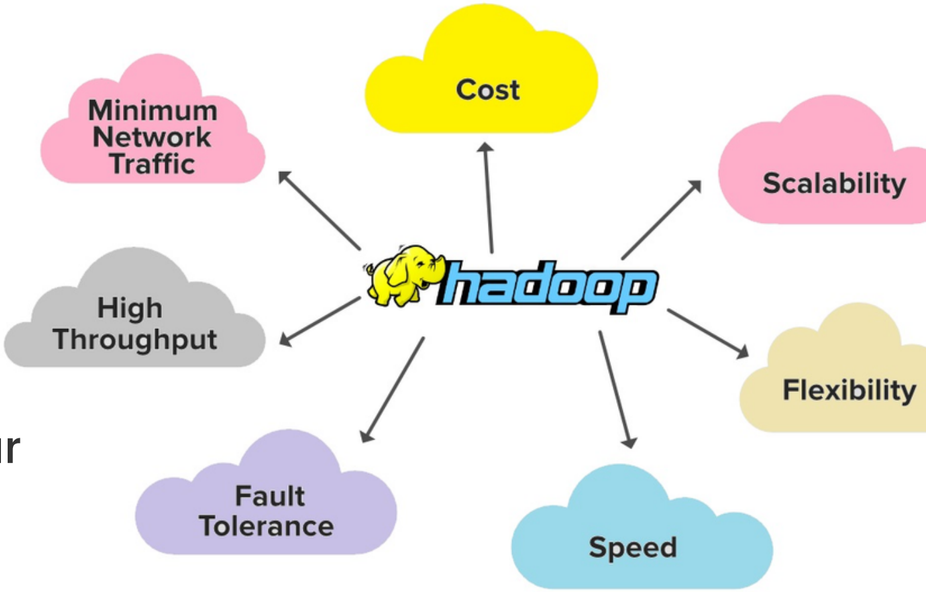
I	Introduction and Aim	Sadikh (21BDS059)
II	HDFS and SPARK Architecture	Srinivas (21BDS012)
III	The Algorithm and Dataset	Prasanna (21BDS052)
IV	Executing the Unified Engine	Zuhair (21BDS060)

Introduction

- Processing and analyzing large datasets efficiently is crucial in today's world.
- Apache Hadoop Distributed File System (HDFS) and Apache Spark are popular solutions for scalable, fast, and fault-tolerant data processing.
- We developed a Spark application to analyze a massive file stored in HDFS using the word count algorithm.

Aim

- The aim of our project is to demonstrate a Spark application that efficiently processes a large file stored in HDFS.
- We will evaluate the application's performance in terms of speed, scalability, and fault tolerance.
- By leveraging the benefits of HDFS and Spark's processing power, our project aims to provide a scalable and fault-tolerant solution for processing large datasets.
- We will measure performance on various cluster sizes, evaluate different Spark configurations, and assess scalability and fault tolerance to gain insights into the potential of HDFS and Spark working together to tackle big data challenges.

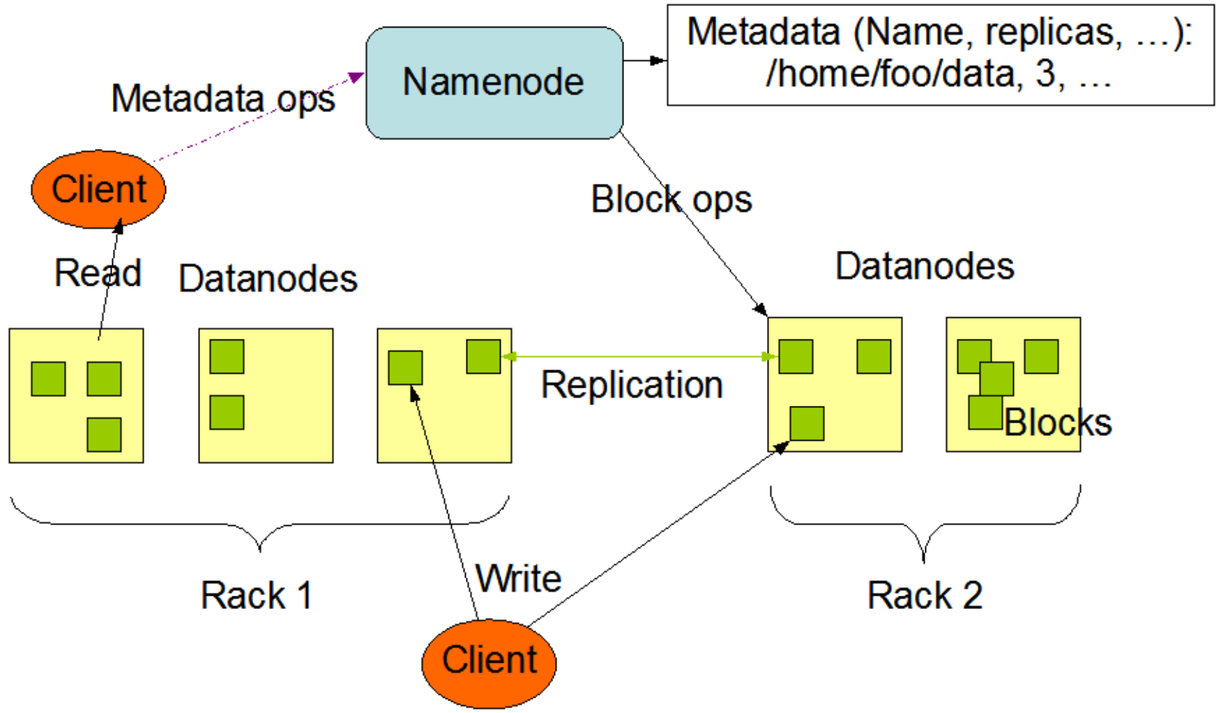


Hadoop Distributive File System Architecture

- HDFS stores large datasets across a cluster for parallel processing, providing fault tolerance and compatibility at a low cost. Apache Spark processes data in parallel and works seamlessly with HDFS for data-intensive applications

- Apache Spark processes large data in parallel and works with HDFS for data-intensive applications, providing scalability, fault tolerance, data locality, and compatibility. Together, they form a powerful distributed computing framework for efficient and cost-effective handling of large datasets.

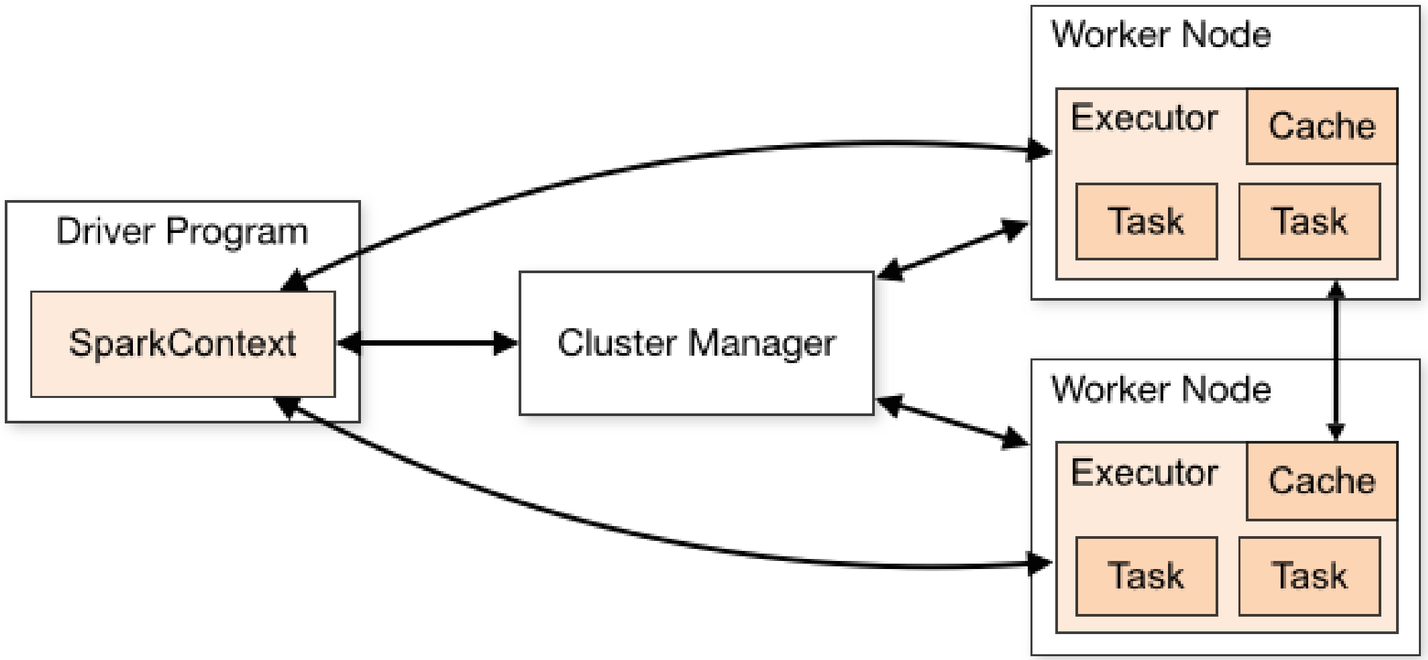
HDFS Architecture



Apache Spark Architecture

- For any algorithm using Spark on an HDFS cluster, data is loaded into HDFS and converted into an RDD, partitioned and assigned to different nodes.

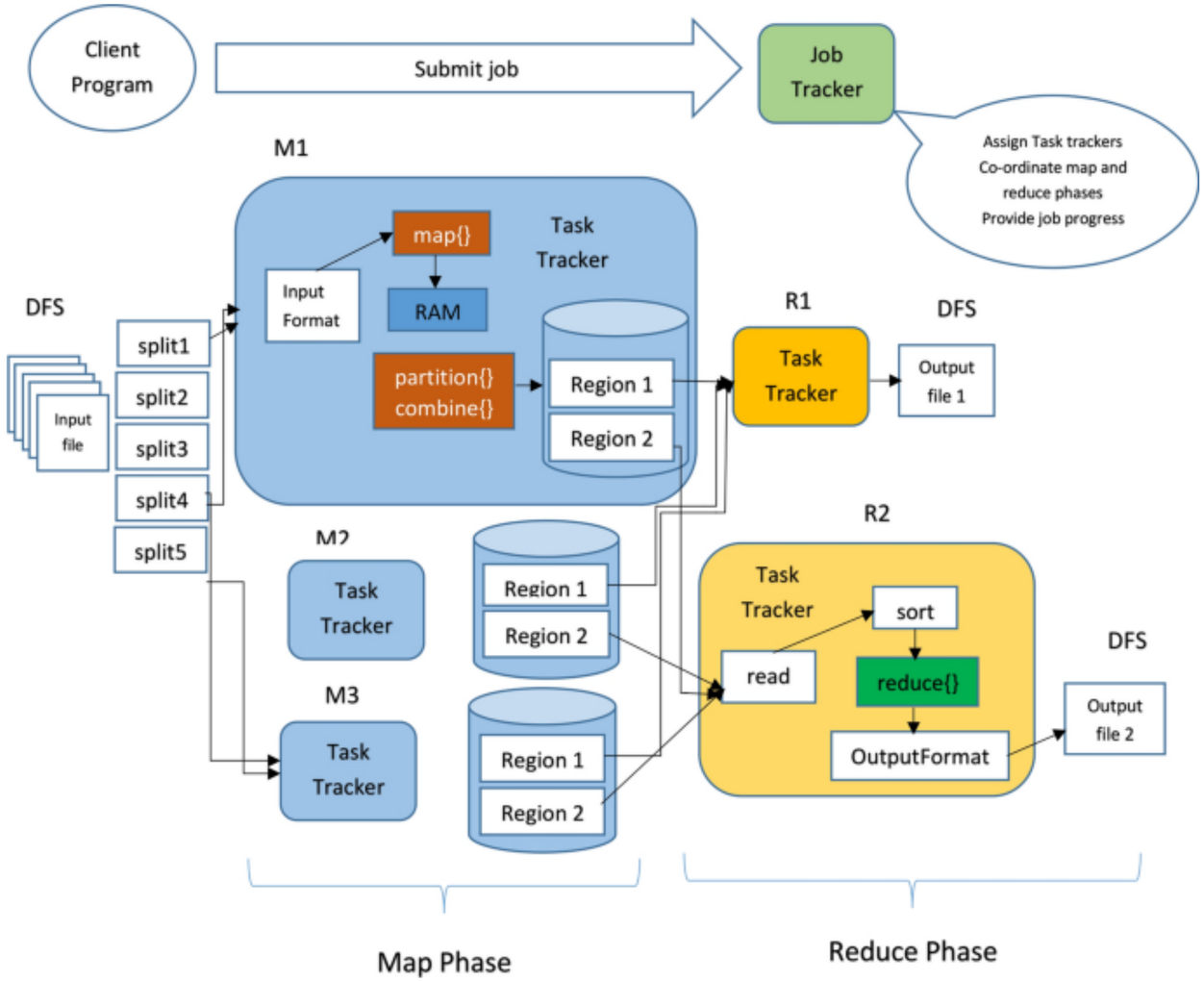
- Spark architecture uses MapReduce paradigm to process data by dividing it into partitions distributed across cluster nodes. Mapping transforms data via "map," reducing aggregates via "reduce" using functions like "reduceByKey" or "aggregateByKey." This approach generates output efficiently in a distributed environment.



Integrating Spark on top of HDFS

Spark and HDFS provide a powerful solution for processing big data. HDFS enables storing and processing large datasets across a cluster, while Spark offers fast and efficient processing. Together, they form a scalable and fault-tolerant framework for extracting insights from vast amounts of data.

As a result, these two technologies complement each other to provide a scalable and fault-tolerant framework for processing vast amounts of data. Overall, this powerful combination enables organizations to extract insights from their data efficiently and effectively.



Algorithm and Dataset

- The algorithm we have chose to run is the classical example for explaining mapreduce execution, The word count algorithm which is written in Java
- The Data set is a text file over 100gb containing all the URLs and logs of web page requests and outputs
- A normal machine takes over 12 hrs to open in the file alone and in some cases they may run into errors or might crash as well
- To process such files we use the sophistication of distributed computing provided by Spark

```java
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.functions

public class WordCount {


    private SparkSession spark;


    public WordCount(SparkSession spark) { this.spark = spark; }


    public void run() {

     Dataset<Row> textFile = spark.read().text("hdfs://10.0.5.215:9000/user/data");

     Dataset<Row> words = textFile.select(functions.explode(functions.split(textFile.col("value")," ")).alias("word"));

     Dataset<Row> wordCounts = words.groupBy("word").count();

     wordCounts.show();
    }


    public static void main(String[] args) {
        SparkSession spark = SparkSession.builder()
                .appName("Word Count Example")
                .master("spark://10.0.5.215:7077")
                .getOrCreate();

        WordCount wordCount = new WordCount(spark);
        wordCount.run();

        spark.stop();
    }
}
```

Executing of the unified Engine : Conclusion

- Spark was deployed in this study to tally unique URLs in a vast dataset of URL access counts. The input data was a preprocessed text file of over 100GB stored in HDFS. However, raising the number of nodes in the Spark cluster failed to considerably enhance the algorithm's performance due to factors such as data locality, CPU utilization, memory, and network bandwidth.

- List of unique URLs and their respective counts generated by the algorithm can aid in website user behavior analysis. The project's methodology is transferable to other Spark-based large dataset projects. It's crucial to account for the limitations of both the algorithm and the Spark cluster while interpreting the outcomes.

- Our findings suggest that a high-performance network infrastructure with high bandwidth and low latency can significantly improve the algorithm's performance. To achieve the best possible results, it is recommended to optimize the network infrastructure for high performance and low latency, along with considering other factors such as memory and CPU utilization when working with large datasets in Spark.

Thank you!