

Ministry of Higher Education and Scientific Research

University of Hassiba Benbouali Chlef

Faculty of Exact & Computer Science

Computer Science Department



# Report

## FINAL TP BIG DATA

Specialization : Computer Science

By

**SID AHMED ABDELALI  
MOUATH BOUMARAF**

Theme :

**Smart Shopping**

Presented on 21/05/2024 before the jury composed of:

Allali Abdelmadjid

Grade/ UHB-Chlef

Président

Academic Year 2024-2025

# Introduction to E-commerce

E-commerce, or electronic commerce, refers to the buying and selling of goods and services over the internet. It involves the use of electronic platforms, such as websites, mobile applications, and social media, to conduct transactions between businesses and consumers or between businesses. E-commerce has revolutionized the way businesses operate and has created new opportunities for entrepreneurs and consumers alike.

E-commerce has continued to grow in popularity and importance, particularly in light of the COVID-19 pandemic, which has led to a surge in online shopping as people seek to minimize in-person interactions.

## The Data Problem in E-commerce

E-commerce websites generate massive amounts of data daily from customer transactions, product listings, user behavior, and seller activities.

The problem is :

- Traditional systems may struggle with this large of data. What is the best solution to storage, handle and process this amount of data ?
- how we can managing and processing this amount of data ?

## Our Solution

The solution is Smart Shopping model uses technology to improve the shopping experience by analyzing large datasets to provide personalized recommendations, dynamic pricing, and real-time inventory updates. In e-commerce, data from sellers and their products is vast and requires powerful tools to process effectively. Hadoop, with its distributed storage and MapReduce processing, enables businesses to handle this data efficiently, extract insights, and enhance decision-making, making it a crucial tool for modern Smart Shopping platforms.

# Hadoop Overview

Hadoop is an open-source framework designed to store and process large datasets efficiently. It uses a distributed system that spreads data across multiple machines, ensuring scalability and fault tolerance. Hadoop is widely used for big data applications due to its ability to handle massive amounts of structured and unstructured data.

## Key Components of Hadoop

1. **HDFS (Hadoop Distributed File System):** A distributed storage system that divides data into blocks and stores them across multiple nodes, ensuring data redundancy and reliability.
2. **MapReduce:** A processing model that breaks tasks into smaller subtasks, processes them in parallel, and combines the results to produce output.
3. **YARN (Yet Another Resource Negotiator):** Manages and schedules resources for data processing tasks.
4. **Hadoop Common:** Provides essential utilities and libraries required for other Hadoop modules.

## How Hadoop Works

1. Data is divided into smaller chunks and stored across a cluster using HDFS.
2. MapReduce processes the data by distributing tasks across nodes, running them in parallel, and aggregating the results.
3. YARN ensures efficient resource allocation and task management during the process.

This design allows Hadoop to handle large datasets efficiently and reliably, making it a cornerstone of big data technologies.

## Overview of HDFS

HDFS (Hadoop Distributed File System) is the storage system used in Hadoop to manage large datasets. It is designed to store data across multiple machines in a distributed manner, ensuring reliability and scalability.

## Key Features of HDFS

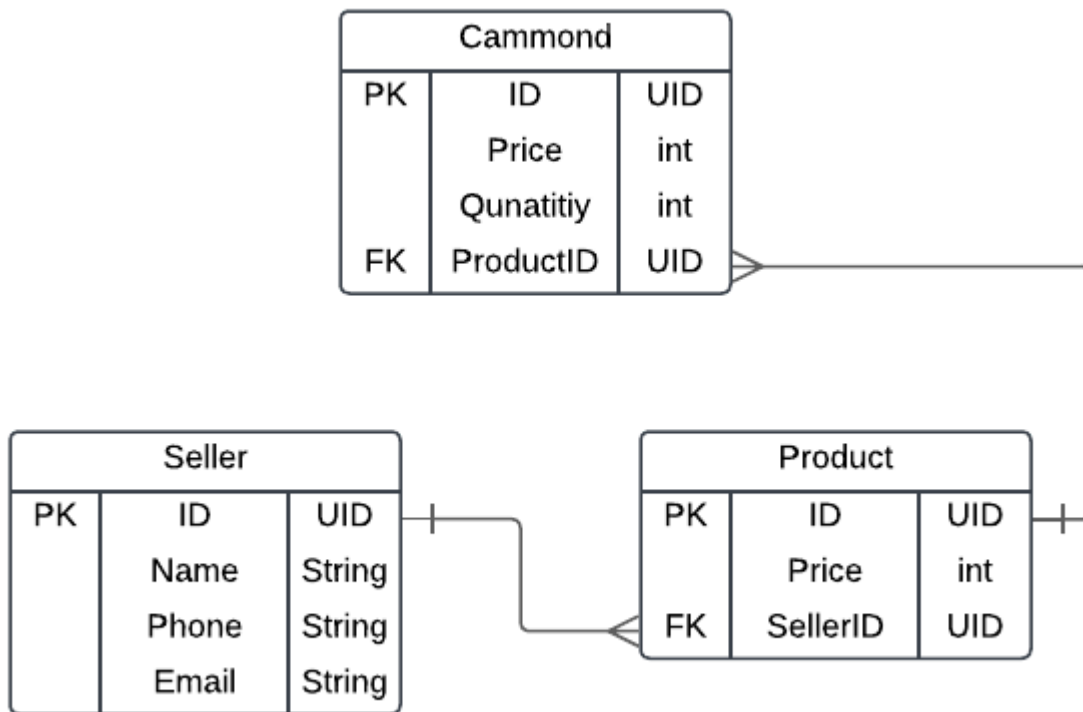
1. **Distributed Storage:** Data is split into blocks and stored across multiple nodes in the cluster.
2. **Fault Tolerance:** Copies (replicas) of data blocks are kept on different nodes, so data remains safe even if a node fails.
3. **Scalability:** It can handle large amounts of data by adding more nodes to the cluster.
4. **High Throughput:** Designed for batch processing, it supports fast data access for big data applications.

## How HDFS Works

1. **NameNode:** The master node that manages metadata (file names, block locations, etc.).
2. **DataNodes:** Worker nodes that store actual data blocks.
3. When a file is uploaded, HDFS divides it into blocks and distributes them across DataNodes. The NameNode keeps track of where each block is stored.

HDFS is the foundation of Hadoop's ability to handle and store big data efficiently.

## Database Design



## Hadoop Usage

### Smart Shopping Data Model

```

date,sellerid,sellername,productid,productname,price,quantity.
2025-01-01,1,Seller_1,P1_1,Product_1_1,48.69,7
2025-01-01,1,Seller_1,P1_2,Product_1_2,50.19,16
2025-01-01,1,Seller_1,P1_3,Product_1_3,59.07,1
2025-01-01,1,Seller_1,P1_4,Product_1_4,85.0,9
  
```

After saving your data, you should run these two commands :

```
hdfs dfs -mkdir /smartshopping
```

This command creates a new directory named `/smartshopping` in the Hadoop Distributed File System (HDFS).

- `hdfs` : Refers to the Hadoop command-line interface.

- `dfs` : Stands for distributed file system, indicating you're interacting with HDFS.
- `mkdir` : The option to create a directory.
- `/smartshopping` : The path of the directory being created in HDFS.

**Result:** A directory `/smartshopping` will be created in the root of the HDFS.

---

```
hdfs dfs -put data.csv /smartshopping
```

This command uploads a file or folder named `data` from the local file system to the `/smartshopping` directory in HDFS

- `hdfs` : Refers to the Hadoop command-line interface.
- `dfs` : Indicates interaction with HDFS.
- `put` : The option to upload files from the local system to HDFS.
- `data` : The local file or folder to be uploaded.
- `/smartshopping` : The destination directory in HDFS.

**Result:** The `data` file or folder from the local file system will be copied to the `/smartshopping` directory in HDFS.

---

## Smart Shopping MapReduce

### Mapper Class

```

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class SalesMapper extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    protected void map(LongWritable key, Text value, Context context) throws
        IOException, InterruptedException {
        String line = value.toString();

        // Skip the header line
        if (line.startsWith("date")) return;

        String[] fields = line.split(",");

        // Extract relevant fields
        String sellerId = fields[1];
        String sellerName = fields[2];
        String productId = fields[3];
        String productName = fields[4];
        double price = Double.parseDouble(fields[5]);
        int quantityBought = Integer.parseInt(fields[6]);
        double revenue = price * quantityBought;

        // Emit: sellerId as key, and sellerName, productId, productName, revenue,
        // quantity as value
        String valueOut = String.join("\t", sellerName, productId, productName,
            String.valueOf(revenue), String.valueOf(quantityBought));
        context.write(new Text(sellerId), new Text(valueOut));
    }
}

```

## Reducer Class

```

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class SalesReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws
        IOException, InterruptedException {
        String sellerName = "";
        double totalRevenue = 0.0;
        String maxProductId = "";
        String maxProductName = "";
        int maxQuantity = 0;

        // Iterate through values for a seller
        for (Text value : values) {
            String[] fields = value.toString().split("\t");
            sellerName = fields[0];
            String productId = fields[1];
            String productName = fields[2];
            double revenue = Double.parseDouble(fields[3]);
            int quantityBought = Integer.parseInt(fields[4]);

            // Update total revenue
            totalRevenue += revenue;

            // Check for the product with the highest quantity
            if (quantityBought > maxQuantity) {
                maxQuantity = quantityBought;
                maxProductId = productId;
                maxProductName = productName;
            }
        }

        // Emit the seller ID, total revenue, and product with max quantity
        String result = String.format("%s\t%.2f\t%s\t%s\t%d", sellerName,
            totalRevenue, maxProductId, maxProductName, maxQuantity);
        context.write(key, new Text(result));
    }
}

```

## Driver Class



```

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class SalesDriver {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: SalesDriver <input path> <output path>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Sales Analysis");

        job.setJarByClass(SalesDriver.class);
        job.setMapperClass(SalesMapper.class);
        job.setReducerClass(SalesReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

return go(f, seed, [])
}

```

```

javac -classpath $(hadoop classpath) -d . SalesMapper.java SalesReducer.java
SalesDriver.java

```

- Compiles the `SalesMapper.java`, `SalesReducer.java`, and `SalesDriver.java` files.
- Ensures that all Hadoop-related libraries are included in the compilation by setting the correct classpath.
- Stores the compiled `.class` files in the current directory.

```
jar -cvf sales_count.jar -C sales_classes/ .
```

- It creates a new JAR file named `sales_count.jar`.
- It includes all files from the `sales_classes/` directory into the JAR.
- The files in the JAR will be organized with their original paths from the `sales_classes/` directory.

```
hadoop jar sales_count.jar SalesDriver /smartshopping/daily_sa
```

- Runs the MapReduce job contained in the `sales_count.jar` file.
- The job processes the input data ( `/smartshopping/daily_sales_data.json` ), using the `SalesMapper` and `SalesReducer` classes defined in the `SalesDriver`.
- The results of the job will be written to the output directory `/smartshopping/output`.

## Result

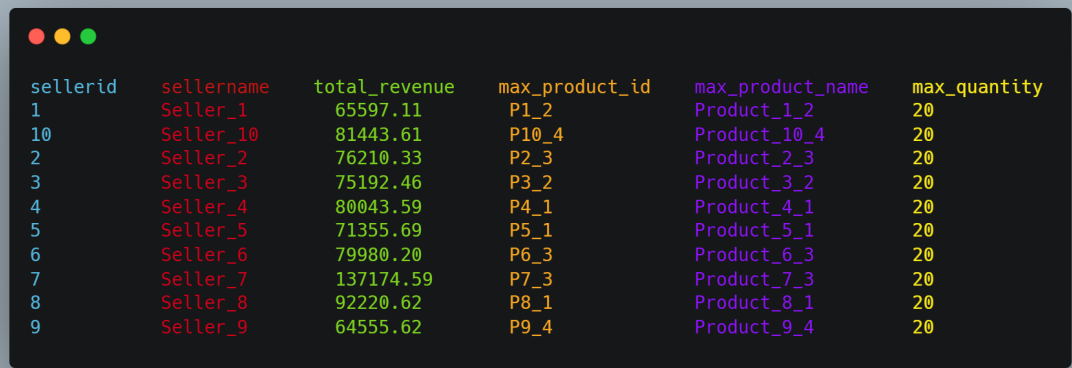
```
hdfs dfs -cat /smartshopping/output/part-00000
```

This command will show the results produced by the MapReduce job stored in the

`/smartshopping/output/` directory.

- `hdfs dfs` : This is the command to interact with the Hadoop Distributed File System (HDFS).
- `cat` : This option displays the content of the specified file(s) in HDFS.
- `/smartshopping/output/part-*` : After running a MapReduce job, the output is typically stored in multiple files prefixed with `part-`. The `*` wildcard means it will display all files that match the pattern (e.g., `part-00000`, `part-00001`, etc.).

## Output



sellerid	sellername	total_revenue	max_product_id	max_product_name	max_quantity
1	Seller_1	65597.11	P1_2	Product_1_2	20
10	Seller_10	81443.61	P10_4	Product_10_4	20
2	Seller_2	76210.33	P2_3	Product_2_3	20
3	Seller_3	75192.46	P3_2	Product_3_2	20
4	Seller_4	80043.59	P4_1	Product_4_1	20
5	Seller_5	71355.69	P5_1	Product_5_1	20
6	Seller_6	79980.20	P6_3	Product_6_3	20
7	Seller_7	137174.59	P7_3	Product_7_3	20
8	Seller_8	92220.62	P8_1	Product_8_1	20
9	Seller_9	64555.62	P9_4	Product_9_4	20