Robert Gordon University

Big Data Programming – CMM705

Course Work Report

MSc in Big Data Analytics

Semester 1

Submitted by:

S.Z. Raeesul Islam

20232953 / 2409649

Table of Contents

Contents

Task 1 - Solution Architecture

1. System Diagram for NBA Basketball

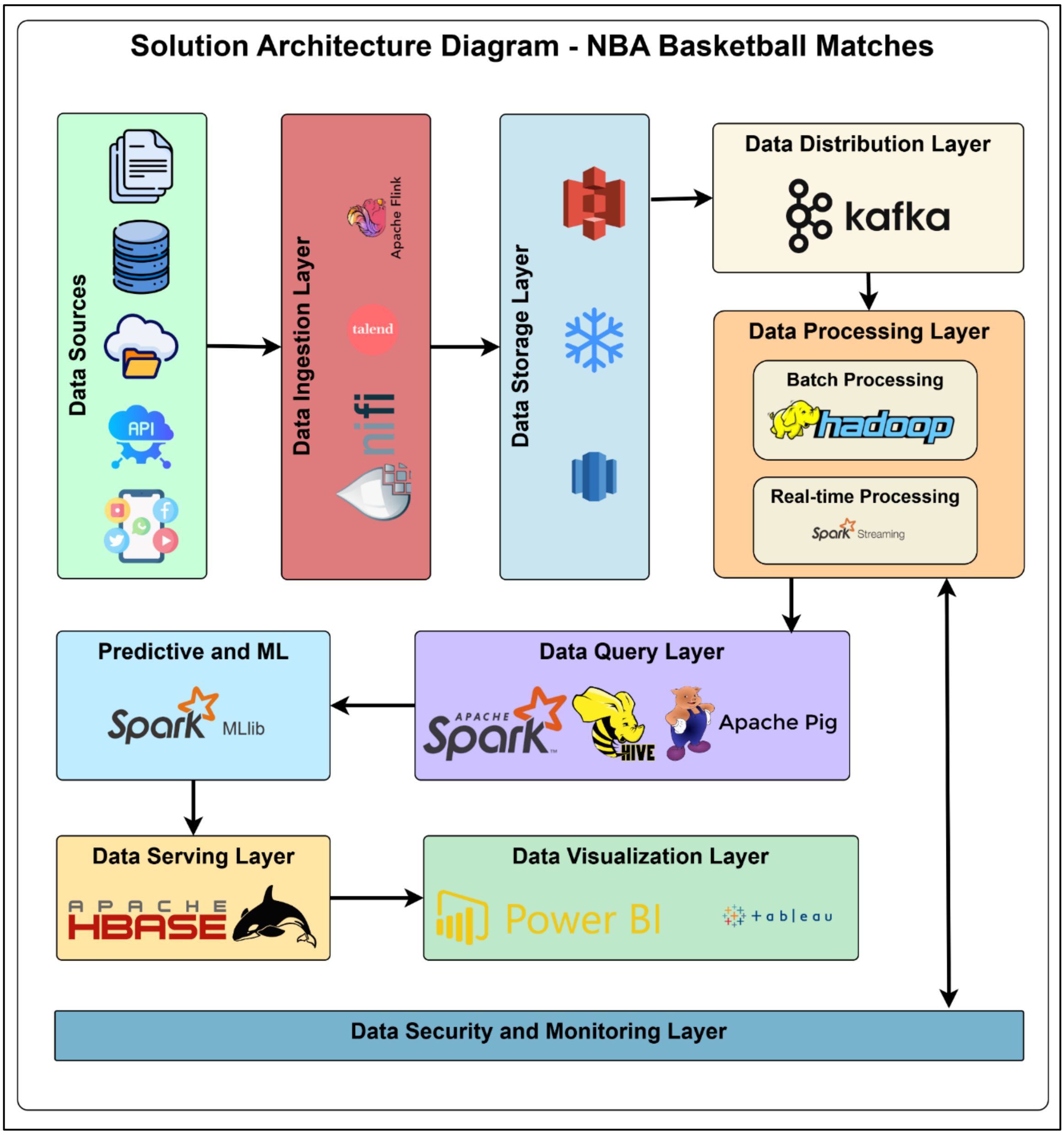


Figure 1. System Diagram

2. Diagram Overview and Role of Components

This **Big Data Analytics** solution for **real-time and historical NBA basketball** matches examines information suited for the NBA. The architecture integrates **key components** in various layers to allow seamless **ingestion of data, storage scalability, robust processing, interactive querying, and insightful visualization.** Modern real-time data streams coupled with batch processing of historical datasets are handled in the system; **predictive analytics and interactive exploration** are supported. Each layer has been optimized to ensure that the **functional requirements** of **real-time analytics and batch processing** are supported, and the **non-functional requirements** of **scalability, availability, and security** are satisfied. What follows below is a detailed description of the role of each layer.

1. **Data Sources Layer:** The layer is the source location of data, which could be from APIs for live game feeds, social media platforms for sentiment, cloud databases for structured historical data, and local files for offline data ingestion. Its main purpose is to deliver a reliable and continuous supply of real-time and batch data to the ingestion process. Scalability remains the focal point here; the architecture must allow for the addition of new data streams or sources without disruption.
2. **Data Ingestion Layer:** This layer is concerned with the collection and transportation of raw data from various sources into the system. The tools used are Apache NiFi, Apache Flink, Talend. This layer ensures that the data is in the correct format and ready for processing, and manages high throughput, and low latency to meet real-time analytics requirements.
3. **Data Distribution Layer:** This layer for distribution of data employs Apache Kafka as a message broker to enable efficient and real-time message flow between ingestions and processing. It buffers incoming data, provides protection from faults and provides scalable low-latency data distributing thus making it possible for multiple consumers to subscribe to data streams.
4. **Data Storage Layer:** This layer is designed for both short-term and long-term data storage; HDFS for large-scale batch storage, Amazon Redshift/ Snowflake for structured data analysis, and HBase for real-time data storage, in a way that makes it scalable, secure, and reliable throughout the system.
5. **Data Processing Layer:** This layer takes care of load, such as real-time and batch, with its main responsibilities of real-time processing of date data for immediate insights and handling of Hadoop batch jobs for deep analysis. It is designed to ensure efficient processing on transformation, aggregation and feature engineering so that they can support real-time dashboard needs and periodic analytical reports.
6. **Data Query Layer:** This Layer is important in data exploration, facilitating interactive queries, such as SQL-like on structured data built on advanced transformations for semi-structured datasets, thus enabling decision making and exploration at speed by analysts and engineers.
7. **Predictive Analytics** is otherwise done by the Spark MLlib library for model building and training, for example game outcome predictions, anomaly detections, trend forecasting, etc. thus providing a shift from descriptive analytics to predictive insights driving decisions with data.
8. **Data Serving Layer:** Apache HBase represents this layer, which provides fast access to the already processed data for the applications and dashboards downstream. It allows for low latency read/write operations, serving efficiently both visualization tools and user queries.

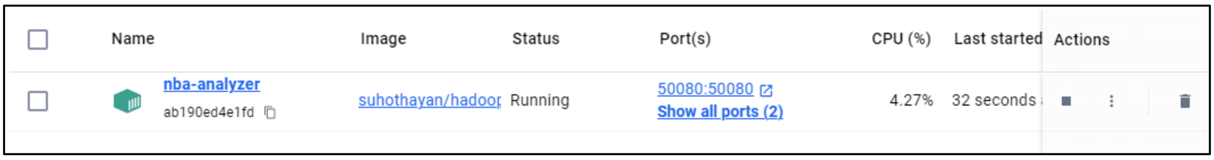
9. **Data Visualization Layer**: with the help of tools like Power BI and Tableau, enables the real-time follow-up of facilitating metrics, such as points and player statistics, making actionable information lucidly understood and available for decision-makers.
10. **Data Security and Monitoring Layer:** This layer ensures a system-wide practice of security, monitoring, encryption, and role-based access control, in addition to pipeline monitoring tools, to ensure that data is kept intact, compliant, and operationally continuous.

Task 2 - Data Analysis

The first step involves mounting the local machine's data set to create a Docker container. The image **suhothayan/hadoop-spark-pig-hive:2.9.2** is used.

```
docker run -it -p 8081:8081 -p 50080:50080 --name nba-analyzer -v
C:\Users\HP\Desktop\NBA:/resource -d suhothayan/hadoop-spark-pig-
hive:2.9.2
```

```
PS C:\Users\HP> docker run -it -p 8081:8081 -p 50080:50080 --name nba-analyzer -v C:\Users\HP\Desktop\NBA:/resource -d s
uhothayan/hadoop-spark-pig-hive:2.9.2
ab190ed4e1fdfd6d735a6b46f344e4e15269c40a477584e9ea166ed02045b58c
PS C:\Users\HP> docker ps
CONTAINER ID   IMAGE                                    COMMAND                  CREATED          STATUS          PORTS
                                                                                                                   NAMES
ab190ed4e1fd   suhothayan/hadoop-spark-pig-hive:2.9.2   "/etc/bootstrap.sh -d"   10 seconds ago   Up 6 seconds    8030-80
33/tcp, 8040/tcp, 8042/tcp, 8080/tcp, 8088/tcp, 9000/tcp, 13562/tcp, 40661/tcp, 50010/tcp, 50020/tcp, 50070/tcp, 0.0.0.0
:8081->8081/tcp, 50075/tcp, 50090/tcp, 0.0.0.0:50080->50080/tcp   nba-analyzer
PS C:\Users\HP>
```



Get into the docker container to view the mounted resource.

```
docker exec -it nba-analyzer bash
```

```
PS C:\Users\HP> docker exec -it nba-analyzer bash
root@ab190ed4e1fd:/# ls
bin   derby.log  etc   lib    media          mnt  proc      root  sbin  sys  usr
boot  dev        home  lib64  metastore_db   opt  resource  run   srv   tmp  var
root@ab190ed4e1fd:/# cd resource/
root@ab190ed4e1fd:/resource# ls
NBA-data.csv
root@ab190ed4e1fd:/resource#
```

The data set must be added to the HDFS Folder to complete the task. First, make a directory in HDFS, and then transfer the file to it.

```
hdfs dfs -mkdir nba-data
hdfs dfs -put NBA-data.csv nba-data/NBA-data.csv
hdfs dfs -ls nba-data
```

```
root@ab190ed4e1fd:/resource# hdfs dfs -mkdir nba-data
root@ab190ed4e1fd:/resource# hdfs dfs -ls
Found 2 items
drwxr-xr-x   - root supergroup          0 2019-07-21 16:09 input
drwxr-xr-x   - root supergroup          0 2024-12-14 07:32 nba-data
root@ab190ed4e1fd:/resource# hdfs dfs -put NBA-data.csv nba-data/NBA-data.csv
root@ab190ed4e1fd:/resource# hdfs dfs -ls nba-data
Found 1 items
-rw-r--r--   1 root supergroup   81584756 2024-12-14 07:32 nba-data/NBA-data.csv
root@ab190ed4e1fd:/resource#
```

1. Hadoop MapReduce

1. Most Scoring Quarter Analysis

MostScoringQuarterAnalysis.java to obtain the most scoring quarters for each team during various game periods in the entire tournament, a Java class is developed with a mapper, reducer, and main function.

Mapper function

```
public class MostScoringQuarterAnalysis {
    public static class MostScoringQuarterMapper extends Mapper<LongWritable, Text, Text, Text> {  1 usage

        @Override
        protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

            Map<String, String> monthMapping = getMonthMapping();

            // Split the line into different fields.
            String[] values = value.toString().split( regex: ",");
            if (values.length < 25) return; // Skip invalid lines

            // extract the necessary fields
            String Score = values[24].trim();
            String GameID = values[2].trim();
            String Event = values[1].trim();
            String Period = values[5].trim();
            String Team = values[11].trim();

            // Fixing the invalid scores values like "02-Feb"
            if (Score.matches( regex: "\\d{1,2}-[a-zA-Z]{3}")) {
                String[] parts = Score.split( regex: "-");
                String day = parts[0];
                String month = monthMapping.getOrDefault(parts[1],  defaultValue: "");
                if (!month.isEmpty()) {
                    Score = day + "-" + month;
                } else {
                    return; // Invalid score
```

```
public class MostScoringQuarterAnalysis {
    public static class MostScoringQuarterMapper extends Mapper<LongWritable, Text, Text, Text> {  1 usage
        protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
                }
            } else if (Score.matches( regex: "[a-zA-Z]{3}-\\d{2}")) {
                String[] parts = Score.split( regex: "-");
                String month = monthMapping.getOrDefault(parts[0],  defaultValue: "");
                String day = parts[1];
                if (!month.isEmpty()) {
                    Score = month + "-" + day;
                } else {
                    return; // Invalid score
                }
            }

            // Filter invalid records
            if (!Score.contains("-") || Team.isEmpty() || Score.isEmpty()) {
                return;
            }

            // Emit the GAME_ID as the key and the processed record as the value
            String record = String.join( delimiter: ",", Event, Period, Team, Score);
            context.write(new Text(GameID), new Text(record));
        }

        private Map<String, String> getMonthMapping() {  1 usage
            Map<String, String> monthMapping = new HashMap<>();
            monthMapping.put("Jan", "1");
```

```java
public class MostScoringQuarterAnalysis {
    public static class MostScoringQuarterMapper extends Mapper<LongWritable, Text, Text, Text> {  1 usage
        protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            }

            // Emit the GAME_ID as the key and the processed record as the value
            String record = String.join( delimiter: ",", Event, Period, Team, Score);
            context.write(new Text(GameID), new Text(record));
        }

        private Map<String, String> getMonthMapping() {  1 usage
            Map<String, String> monthMapping = new HashMap<>();
            monthMapping.put("Jan", "1");
            monthMapping.put("Feb", "2");
            monthMapping.put("Mar", "3");
            monthMapping.put("Apr", "4");
            monthMapping.put("May", "5");
            monthMapping.put("Jun", "6");
            monthMapping.put("Jul", "7");
            monthMapping.put("Aug", "8");
            monthMapping.put("Sep", "9");
            monthMapping.put("Oct", "10");
            monthMapping.put("Nov", "11");
            monthMapping.put("Dec", "12");
            return monthMapping;
        }
    }
```

Reducer function

```java
public class MostScoringQuarterAnalysis {
    public static class MostScoringQuarterReducer extends Reducer<Text, Text, Text, Text> {  1 usage

        private final Map<String, Map<Integer, Integer>> globalTeamPeriodScores = new HashMap<>();  3 usages

        @Override
        protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            // Key: gameID
            // Values: list of "eventNum, quarter, team, scoreboard"

            List<String[]> plays = new ArrayList<>();

            for (Text value : values) {
                String[] parts = value.toString().split( regex: ",");
                plays.add(parts);
            }

            // Sort the plays by eventNum
            plays.sort(Comparator.comparingInt( String[] p -> Integer.parseInt(p[0])));

            Map<String, Map<Integer, Integer>> teamPeriodScores = new HashMap<>();

            String previousScore = null;

            for (String[] parts : plays) {
                int eventNum = Integer.parseInt(parts[0]);
                int period = Integer.parseInt(parts[1]);
                String scoringTeam = parts[2];
```

```java
public class MostScoringQuarterAnalysis {
    public static class MostScoringQuarterReducer extends Reducer<Text, Text, Text, Text> {
        protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
                String scoringTeam = parts[2];
                String scoreboard = parts[3];

                int playScore;

                if (previousScore == null) {
                    playScore = Integer.parseInt(scoreboard.split("-")[1].trim());
                } else {
                    String[] currentScoreParts = scoreboard.split("-");
                    String[] previousScoreParts = previousScore.split("-");

                    int currentScore = Integer.parseInt(currentScoreParts[0].trim()) + Integer.parseInt(currentScoreParts[1].trim());
                    int previousScoreValue = Integer.parseInt(previousScoreParts[0].trim()) + Integer.parseInt(previousScoreParts[1].trim());

                    playScore = currentScore - previousScoreValue;
                }

                teamPeriodScores.putIfAbsent(scoringTeam, new HashMap<>());
                Map<Integer, Integer> quarterScores = teamPeriodScores.get(scoringTeam);
                quarterScores.put(period, quarterScores.getOrDefault(period, 0) + playScore);

                previousScore = scoreboard;
            }

            for (Map.Entry<String, Map<Integer, Integer>> teamEntry : teamPeriodScores.entrySet()) {
```

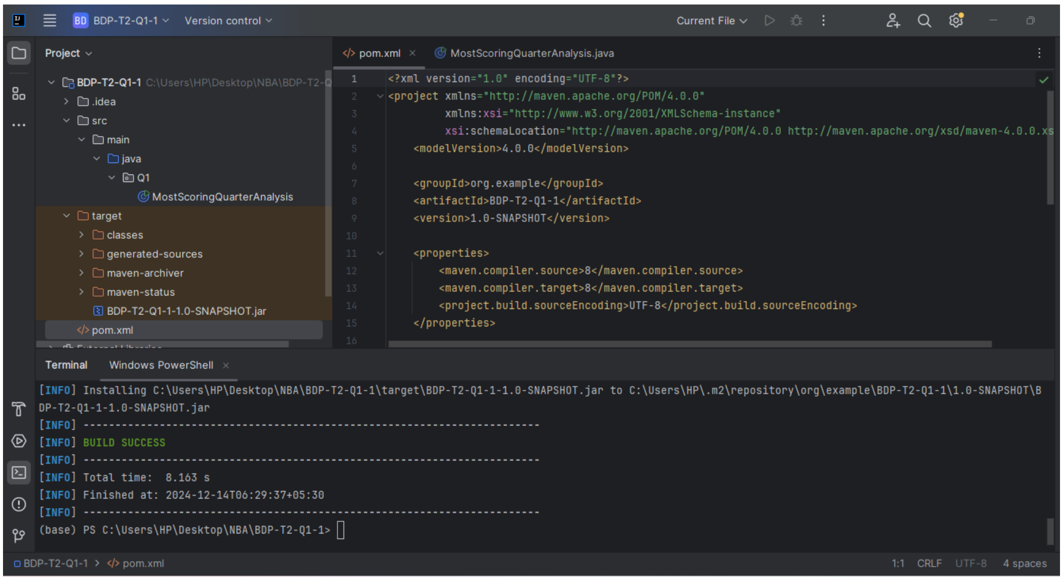
```java
public class MostScoringQuarterAnalysis {
    public static class MostScoringQuarterReducer extends Reducer<Text, Text, Text, Text> {
        protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            for (Map.Entry<String, Map<Integer, Integer>> teamEntry : teamPeriodScores.entrySet()) {
                String team = teamEntry.getKey();
                Map<Integer, Integer> quarterScores = teamEntry.getValue();

                globalTeamPeriodScores.putIfAbsent(team, new HashMap<>());
                Map<Integer, Integer> globalQuarterScores = globalTeamPeriodScores.get(team);

                for (Map.Entry<Integer, Integer> quarterEntry : quarterScores.entrySet()) {
                    int quarter = quarterEntry.getKey();
                    int score = quarterEntry.getValue();
                    globalQuarterScores.put(quarter, globalQuarterScores.getOrDefault(quarter, 0) + score);
                }
            }
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            for (Map.Entry<String, Map<Integer, Integer>> teamEntry : globalTeamPeriodScores.entrySet()) {
                String team = teamEntry.getKey();
                Map<Integer, Integer> quarterScores = teamEntry.getValue();

                int mostScoringPeriod = -1;
                int mostScore = -1;
```

```java
public class MostScoringQuarterAnalysis {
    public static class MostScoringQuarterReducer extends Reducer<Text, Text, Text, Text> {
        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            for (Map.Entry<String, Map<Integer, Integer>> teamEntry : globalTeamPeriodScores.entrySet()) {
                String team = teamEntry.getKey();
                Map<Integer, Integer> quarterScores = teamEntry.getValue();

                int mostScoringPeriod = -1;
                int mostScore = -1;

                for (Map.Entry<Integer, Integer> quarterEntry : quarterScores.entrySet()) {
                    int quarter = quarterEntry.getKey();
                    int score = quarterEntry.getValue();

                    if (score > mostScore) {
                        mostScore = score;
                        mostScoringPeriod = quarter;
                    }
                }

                context.write(new Text(team), new Text("Most Scoring Period: " + mostScoringPeriod + ", Score: " + mostScore));
            }
        }
    }
```

Main function

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

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Most Scoring Quarter Analysis");
        job.setJarByClass(MostScoringQuarterAnalysis.class);

        job.setMapperClass(MostScoringQuarterMapper.class);
        job.setReducerClass(MostScoringQuarterReducer.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);
    }
}
```

After that, the project is constructed to produce the jar file in the designated folder.



Since the jar file has been created, the YARN command is used to carry out the task.

yarn jar BDP-T2-Q1-1/target/BDP-T2-Q1-1-1.0-SNAPSHOT.jar Q1.MostScoringQuarterAnalysis nba-data/NBA-data.csv output/T02_Q01_1_outputs

```
root@ab190ed4e1fd:/resource# yarn jar BDP-T2-Q1-1/target/BDP-T2-Q1-1-1.0-SNAPSHOT.jar Q1.MostScoringQuarterAnalysis nba-data/NBA-data.csv output/T02_Q01_1_outputs
24/12/14 07:34:24 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
24/12/14 07:34:24 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
24/12/14 07:34:25 INFO input.FileInputFormat: Total input files to process : 1
24/12/14 07:34:25 INFO mapreduce.JobSubmitter: number of splits:1
24/12/14 07:34:25 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publisher.enabled
24/12/14 07:34:25 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1734161408917_0001
24/12/14 07:34:26 INFO impl.YarnClientImpl: Submitted application application_1734161408917_0001
24/12/14 07:34:26 INFO mapreduce.Job: The url to track the job: http://ab190ed4e1fd:8088/proxy/application_1734161408917_0001/
```

The output will be kept in the "`output/ T02_Q01_1_outputs`" folder. It appears as follows:

```
hdfs dfs -ls output/T02_Q01_1_outputs
hdfs dfs -cat output/T02_Q01_1_outputs/part-r-00000
```

```
        File Output Format Counters
                Bytes Written=1270
root@ab190ed4e1fd:/resource# hdfs dfs -ls output/T02_Q01_1_outputs
Found 2 items
-rw-r--r--   1 root supergroup          0 2024-12-14 07:34 output/T02_Q01_1_outputs/_SUCCESS
-rw-r--r--   1 root supergroup       1270 2024-12-14 07:34 output/T02_Q01_1_outputs/part-r-00000
root@ab190ed4e1fd:/resource# hdfs dfs -cat output/T02_Q01_1_outputs/part-r-00000
Hawks   Most Scoring Period: 2, Score: 1882
Cavaliers       Most Scoring Period: 4, Score: 1952
Hornets Most Scoring Period: 2, Score: 1921
76ers   Most Scoring Period: 2, Score: 1987
Lakers  Most Scoring Period: 1, Score: 2126
Celtics Most Scoring Period: 2, Score: 1985
Kings   Most Scoring Period: 1, Score: 2130
Pistons Most Scoring Period: 1, Score: 2003
Warriors        Most Scoring Period: 2, Score: 1906
Nuggets Most Scoring Period: 1, Score: 2070
Wizards Most Scoring Period: 1, Score: 1903
Raptors Most Scoring Period: 1, Score: 1998
Spurs   Most Scoring Period: 2, Score: 2021
Bucks   Most Scoring Period: 2, Score: 2149
Suns    Most Scoring Period: 4, Score: 2030
```

```
Spurs   Most Scoring Period: 2, Score: 2021
Bucks   Most Scoring Period: 2, Score: 2149
Suns    Most Scoring Period: 4, Score: 2030
Grizzlies       Most Scoring Period: 2, Score: 1958
Mavericks       Most Scoring Period: 1, Score: 2122
Magic   Most Scoring Period: 3, Score: 2035
Timberwolves    Most Scoring Period: 1, Score: 2000
Rockets Most Scoring Period: 4, Score: 1998
Nets    Most Scoring Period: 2, Score: 1927
Trail Blazers   Most Scoring Period: 1, Score: 1961
Heat    Most Scoring Period: 3, Score: 1816
Jazz    Most Scoring Period: 4, Score: 1927
SuperSonics     Most Scoring Period: 2, Score: 2164
Bulls   Most Scoring Period: 1, Score: 1858
Pacers  Most Scoring Period: 3, Score: 1893
Clippers        Most Scoring Period: 4, Score: 1953
Knicks  Most Scoring Period: 2, Score: 1903
root@ab190ed4e1fd:/resource#
```

2. Most Scored Player Analysis

MostScoredPlayerAnalysis.java to obtain the total score of the most scored player in the full tournament, a Java class is developed with a mapper, reducer, and main function.

Mapper function

```java
public class MostScoredPlayerAnalysis {
    public static class HighestScoringPlayerMapper extends Mapper<LongWritable, Text, Text, Text> {

        // Month mapping method
        private Map<String, String> getMonthMapping() {
            Map<String, String> monthMapping = new HashMap<>();
            monthMapping.put("Jan", "1");
            monthMapping.put("Feb", "2");
            monthMapping.put("Mar", "3");
            monthMapping.put("Apr", "4");
            monthMapping.put("May", "5");
            monthMapping.put("Jun", "6");
            monthMapping.put("Jul", "7");
            monthMapping.put("Aug", "8");
            monthMapping.put("Sep", "9");
            monthMapping.put("Oct", "10");
            monthMapping.put("Nov", "11");
            monthMapping.put("Dec", "12");
            return monthMapping;
        }

        @Override
        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {

            Map<String, String> monthMapping = getMonthMapping();

            String[] values = value.toString().split(",");
```

```java
public class MostScoredPlayerAnalysis {
    public static class HighestScoringPlayerMapper extends Mapper<LongWritable, Text, Text, Text> {
        protected void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {

            Map<String, String> monthMapping = getMonthMapping();

            String[] values = value.toString().split(",");
            if (values.length < 25) return;

            String score = values[24].trim();
            String gameId = values[2].trim();
            String player = values[7].trim();
            String event = values[1].trim();

            // Normalize score format
            if (score.matches("\\d{1,2}-[a-zA-Z]{3}")) {
                String[] parts = score.split("-");
                String day = parts[0];
                String month = monthMapping.getOrDefault(parts[1], "");
                if (!month.isEmpty()) {
                    score = day + "-" + month;
                } else {
                    return;
                }
            } else if (score.matches("[a-zA-Z]{3}-\\d{2}")) {
                String[] parts = score.split("-");
                String month = monthMapping.getOrDefault(parts[0], "");
```

```java
public class MostScoredPlayerAnalysis {
    public static class HighestScoringPlayerMapper extends Mapper<LongWritable, Text, Text, Text> {
        protected void map(LongWritable key, Text value, Context context)
                    score = day + "-" + month;
                } else {
                    return;
                }
            } else if (score.matches("[a-zA-Z]{3}-\\d{2}")) {
                String[] parts = score.split("-");
                String month = monthMapping.getOrDefault(parts[0], "");
                String day = parts[1];
                if (!month.isEmpty()) {
                    score = month + "-" + day;
                } else {
                    return;
                }
            }

            // Validate record
            if (!score.contains("-") || player.isEmpty() || score.isEmpty()) {
                return;
            }

            // Emit record
            String record = String.join(",", event, player, score);
            context.write(new Text(gameId), new Text(record));
        }
    }
```

Reducer function

```java
public class MostScoredPlayerAnalysis {
    public static class HighestScoringPlayerReducer extends Reducer<Text, Text, Text, Text> {
        // Global map to track player scores
        private final Map<String, Integer> playerScores = new HashMap<>();

        @Override
        protected void reduce(Text key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException {

            // List to store and sort plays
            List<String[]> plays = new ArrayList<>();

            // Collect plays
            for (Text value : values) {
                String[] parts = value.toString().split(",");
                plays.add(parts);
            }

            // Sort plays by event number
            plays.sort(Comparator.comparingInt(p -> Integer.parseInt(p[0])));

            // Track previous score for calculation
            String previousScore = null;

            for (String[] parts : plays) {
                String player = parts[1];
                String currentScoreboard = parts[2];
```

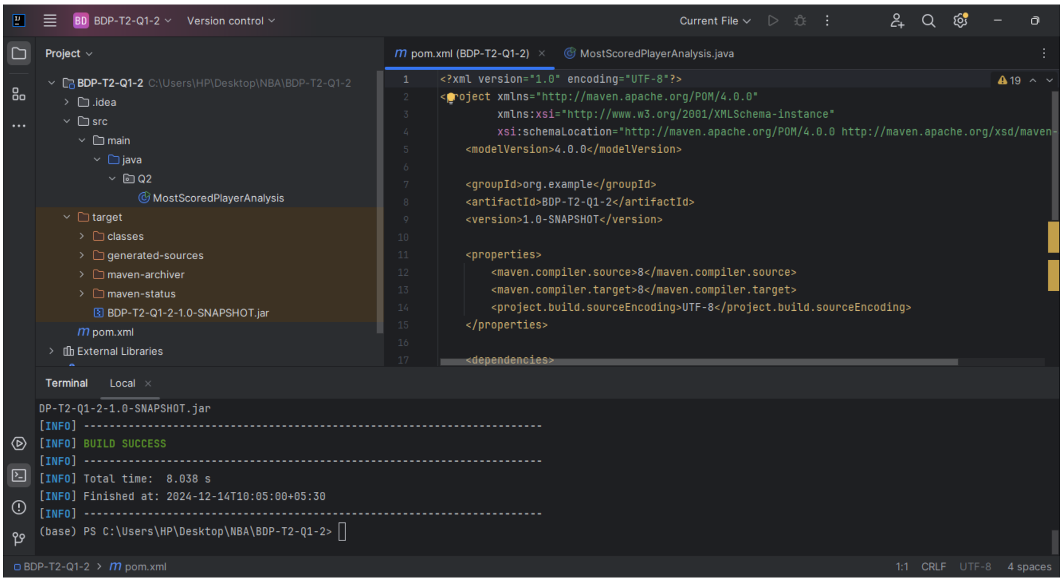
```java
public class MostScoredPlayerAnalysis {
    public static class HighestScoringPlayerReducer extends Reducer<Text, Text, Text, Text> {
        protected void reduce(Text key, Iterable<Text> values, Context context)
                int playScore;

                // Calculate play score
                if (previousScore == null) {
                    playScore = Integer.parseInt(currentScoreboard.split("-")[0].trim()) +
                            Integer.parseInt(currentScoreboard.split("-")[1].trim());
                } else {
                    int currentScore = Integer.parseInt(currentScoreboard.split("-")[0].trim()) +
                            Integer.parseInt(currentScoreboard.split("-")[1].trim());
                    int previousScoreValue = Integer.parseInt(previousScore.split("-")[0].trim()) +
                            Integer.parseInt(previousScore.split("-")[1].trim());

                    playScore = currentScore - previousScoreValue;
                }

                // Update player score
                playerScores.put(player, playerScores.getOrDefault(player, 0) + playScore);
                previousScore = currentScoreboard;
            }
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            // Find the highest-scoring player
```

```java
public class MostScoredPlayerAnalysis {
    public static class HighestScoringPlayerReducer extends Reducer<Text, Text, Text, Text> {

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            // Find the highest-scoring player
            String highestScoringPlayer = null;
            int highestScore = 0;

            for (Map.Entry<String, Integer> entry : playerScores.entrySet()) {
                String player = entry.getKey();
                int totalScore = entry.getValue();

                if (totalScore > highestScore) {
                    highestScore = totalScore;
                    highestScoringPlayer = player;
                }
            }

            // Emit the highest-scoring player
            if (highestScoringPlayer != null) {
                context.write(new Text("Most Scoring Player "),
                        new Text(highestScoringPlayer + ", Total Score: " + highestScore));
            }
        }
    }
```

Main function

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

        // Create configuration
        Configuration conf = new Configuration();

        // Create job instance
        Job job = Job.getInstance(conf,  jobName: "Highest Scoring Player Analysis");
        job.setJarByClass(MostScoredPlayerAnalysis.class);

        // Set Mapper and Reducer
        job.setMapperClass(HighestScoringPlayerMapper.class);
        job.setReducerClass(HighestScoringPlayerReducer.class);

        // Set output key and value types
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // Ensure single reducer for global comparison
        job.setNumReduceTasks(1);

        // Set input and output paths
        FileInputFormat.addInputPath(job, new Path(args[0]));
```

After that, the project is constructed to produce the jar file in the designated folder.



Since the jar file has been created, the YARN command is used to carry out the task.

yarn jar BDP-T2-Q1-2/target/BDP-T2-Q1-2-1.0-SNAPSHOT.jar Q2.MostScoredPlayerAnalysis nba-data/NBA-data.csv output/T02_Q01_2_outputs

```
root@ab190ed4e1fd:/resource# yarn jar BDP-T2-Q1-2/target/BDP-T2-Q1-2-1.0-SNAPSHOT.jar Q2.MostScoredPlayerAnalysis nba-da
ta/NBA-data.csv output/T02_Q01_2_outputs
24/12/14 07:43:15 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
24/12/14 07:43:16 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the To
ol interface and execute your application with ToolRunner to remedy this.
24/12/14 07:43:17 INFO input.FileInputFormat: Total input files to process : 1
24/12/14 07:43:17 INFO mapreduce.JobSubmitter: number of splits:1
24/12/14 07:43:17 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. I
nstead, use yarn.system-metrics-publisher.enabled
24/12/14 07:43:18 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1734161408917_0003
24/12/14 07:43:18 INFO impl.YarnClientImpl: Submitted application application_1734161408917_0003
24/12/14 07:43:18 INFO mapreduce.Job: The url to track the job: http://ab190ed4e1fd:8088/proxy/application_1734161408917
_0003/
```

The output will be kept in the "**`output/ T02_Q01_2_outputs`**" folder. It appears as follows:

`hdfs dfs -ls output/T02_Q01_2_outputs`

`hdfs dfs -cat output/T02_Q01_2_outputs/part-r-00000`

```
        File Output Format Counters
                Bytes Written=57
root@ab190ed4e1fd:/resource# hdfs dfs -ls output/T02_Q01_2_outputs
Found 2 items
-rw-r--r--   1 root supergroup          0 2024-12-14 07:43 output/T02_Q01_2_outputs/_SUCCESS
-rw-r--r--   1 root supergroup         57 2024-12-14 07:43 output/T02_Q01_2_outputs/part-r-00000
root@ab190ed4e1fd:/resource# hdfs dfs -cat output/T02_Q01_2_outputs/part-r-00000
Most Scoring Player     Jerry Stackhouse, Total Score: 2370
root@ab190ed4e1fd:/resource#
```

The most scored player is **"Jerry Stakehouse",** and he scored **2370** points.

2. Apache Hive

The following directories are made before running **Apache Hive** commands. Input data and outputs are stored in these directories.

```
hdfs dfs -mkdir /user/nba_data
hdfs dfs -mkdir /user/hive
hdfs dfs -mkdir /user/hive/nba_data
```

```
root@ab190ed4e1fd:/# hdfs dfs -mkdir /user/nba_data
root@ab190ed4e1fd:/# hdfs dfs -ls /user/nba_data
root@ab190ed4e1fd:/# hdfs dfs -ls /user
Found 2 items
drwxr-xr-x   - root supergroup          0 2024-12-14 08:00 /user/nba_data
drwxr-xr-x   - root supergroup          0 2024-12-14 07:34 /user/root
root@ab190ed4e1fd:/# hdfs dfs -mkdir /user/hive
root@ab190ed4e1fd:/# hdfs dfs -ls /user
Found 3 items
drwxr-xr-x   - root supergroup          0 2024-12-14 08:01 /user/hive
drwxr-xr-x   - root supergroup          0 2024-12-14 08:00 /user/nba_data
drwxr-xr-x   - root supergroup          0 2024-12-14 07:34 /user/root
root@ab190ed4e1fd:/# hdfs dfs -mkdir /user/hive/nba_data
root@ab190ed4e1fd:/# hdfs dfs -ls /user/hive
Found 1 items
drwxr-xr-x   - root supergroup          0 2024-12-14 08:01 /user/hive/nba_data
root@ab190ed4e1fd:/#
```

Then the NBA dataset move to the location **/user/hive/nba_data**

```
hdfs dfs -put /resource/NBA-data.csv /user/nba_data/NBA-data.csv
```

```
root@ab190ed4e1fd:/# hdfs dfs -put /resource/NBA-data.csv /user/nba_data/NBA-data.csv
root@ab190ed4e1fd:/# hdfs dfs -ls /user/nba_data
Found 1 items
-rw-r--r--   1 root supergroup   81584756 2024-12-14 08:02 /user/nba_data/NBA-data.csv
root@ab190ed4e1fd:/#
```

To store the dataset, a new database and table are created.

```
create database nba_db;
show databases;
use nba_db;
```

```
hive> create database nba_db;
OK
Time taken: 0.515 seconds
hive> show databases;
OK
default
nba_db
Time taken: 0.077 seconds, Fetched: 2 row(s)
hive> use nba_db;
OK
Time taken: 0.101 seconds
hive>
```

```
CREATE EXTERNAL TABLE nba_table (EVENTID STRING, EVENTNUM STRING,
HOMEDESCRIPTION STRING, PCTIMESTRING STRING, GAME_ID STRING, PERIOD
STRING, PLAYER1_ID STRING, PLAYER1_NAME STRING,
PLAYER1_TEAM_ABBREVIATION STRING, PLAYER1_TEAM_CITY STRING,
PLAYER1_TEAM_ID STRING, PLAYER1_TEAM_NICKNAME STRING, PLAYER2_ID
STRING, PLAYER2_NAME STRING, PLAYER2_TEAM_ABBREVIATION STRING,
```

```
PLAYER2_TEAM_CITY STRING, PLAYER2_TEAM_ID STRING, PLAYER2_TEAM_NICKNAME
STRING, PLAYER3_ID STRING, PLAYER3_NAME STRING,
PLAYER3_TEAM_ABBREVIATION STRING, PLAYER3_TEAM_CITY STRING,
PLAYER3_TEAM_ID STRING, PLAYER3_TEAM_NICKNAME STRING, SCORE STRING,
SCOREMARGIN STRING, VISITORDESCRIPTION STRING, CLEANED_SCORE STRING,
POINTS_SCORED STRING, WINNER STRING, LOSER STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LOCATION '/user/hive/nba_data';
```

```
hive> CREATE EXTERNAL TABLE nba_table (
    >     EVENTID STRING,
    >     EVENTNUM STRING,
    >     HOMEDESCRIPTION STRING,
    >     PCTIMESTRING STRING,
    >     GAME_ID STRING,
    >     PERIOD STRING,
    >     PLAYER1_ID STRING,
    >     PLAYER1_NAME STRING,
    >     PLAYER1_TEAM_ABBREVIATION STRING,
    >     PLAYER1_TEAM_CITY STRING,
    >     PLAYER1_TEAM_ID STRING,
    >     PLAYER1_TEAM_NICKNAME STRING,
    >     PLAYER2_ID STRING,
    >     PLAYER2_NAME STRING,
    >     PLAYER2_TEAM_ABBREVIATION STRING,
    >     PLAYER2_TEAM_CITY STRING,
    >     PLAYER2_TEAM_ID STRING,
    >     PLAYER2_TEAM_NICKNAME STRING,
    >     PLAYER3_ID STRING,
    >     PLAYER3_NAME STRING,
    >     PLAYER3_TEAM_ABBREVIATION STRING,
    >     PLAYER3_TEAM_CITY STRING,
    >     PLAYER3_TEAM_ID STRING,
    >     PLAYER3_TEAM_NICKNAME STRING,
    >     SCORE STRING,
    >     SCOREMARGIN STRING,
    >     VISITORDESCRIPTION STRING,
    >     CLEANED_SCORE STRING,
    >     POINTS_SCORED STRING,
```

```
    >     PLAYER1_TEAM_ABBREVIATION STRING,
    >     PLAYER1_TEAM_CITY STRING,
    >     PLAYER1_TEAM_ID STRING,
    >     PLAYER1_TEAM_NICKNAME STRING,
    >     PLAYER2_ID STRING,
    >     PLAYER2_NAME STRING,
    >     PLAYER2_TEAM_ABBREVIATION STRING,
    >     PLAYER2_TEAM_CITY STRING,
    >     PLAYER2_TEAM_ID STRING,
    >     PLAYER2_TEAM_NICKNAME STRING,
    >     PLAYER3_ID STRING,
    >     PLAYER3_NAME STRING,
    >     PLAYER3_TEAM_ABBREVIATION STRING,
    >     PLAYER3_TEAM_CITY STRING,
    >     PLAYER3_TEAM_ID STRING,
    >     PLAYER3_TEAM_NICKNAME STRING,
    >     SCORE STRING,
    >     SCOREMARGIN STRING,
    >     VISITORDESCRIPTION STRING,
    >     CLEANED_SCORE STRING,
    >     POINTS_SCORED STRING,
    >     WINNER STRING,
    >     LOSER STRING
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > LOCATION '/user/hive/nba_data';
OK
Time taken: 0.758 seconds
hive>
```

The dataset is loaded into the created table folder.

```
LOAD DATA INPATH 'hdfs:/user/nba_data/NBA-data.csv' OVERWRITE INTO
TABLE nba_table;
ALTER TABLE nba_table SET TBLPROPERTIES ("skip.header.line.count"="1");
```

```
hive> LOAD DATA INPATH 'hdfs:/user/nba_data/NBA-data.csv' OVERWRITE INTO TABLE nba_table;
Loading data to table nba_db.nba_table
OK
Time taken: 1.82 seconds
hive> ALTER TABLE nba_table SET TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 0.149 seconds
hive> SELECT * FROM nba_table LIMIT 5;
OK
0       0       20000001                12:00   1       0                                                       0       0
0       76ers   Knicks
1       1       20000001        Jump Ball Camby vs. Ratliff: Tip to Houston     12:00   1       948     Marcus Camby    N
YK      New York        1610612752      Knicks  689     Theo Ratliff    PHI     Philadelphia    1610612755      76ers   2
75      Allan Houston   NYK     New York        1610612752      Knicks                                  0       76ers   K
nicks
2       2       20000001        MISS Sprewell 6' Jump Shot      11:41   1       84      Latrell Sprewell        NYK     N
ew York 1610612752      Knicks  0                                       689     Theo Ratliff    PHI     Philadel
phia    1610612755      76ers                   Ratliff BLOCK (1 BLK)           0       76ers   Knicks
3       3       20000001                11:40   1       1610612755                                              0       0
76ers Rebound           0       76ers   Knicks
4       4       20000001        Camby S.FOUL (P1.T1)    11:29   1       948     Marcus Camby    NYK     New York        1
610612752       Knicks  0                                       0                                               0
76ers   Knicks
Time taken: 1.991 seconds, Fetched: 5 row(s)
hive>
```

1. Top 5 Teams (Total Number of Points Scored)

```
SELECT PLAYER1_TEAM_NICKNAME, SUM(POINTS_SCORED) AS total_points
FROM nba_table
GROUP BY PLAYER1_TEAM_NICKNAME
ORDER BY total_points DESC
LIMIT 5;
```

```
hive> SELECT PLAYER1_TEAM_NICKNAME, SUM(POINTS_SCORED) AS total_points
    > FROM nba_table
    > GROUP BY PLAYER1_TEAM_NICKNAME
    > ORDER BY total_points DESC
    > LIMIT 5;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different
execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20241214085429_6164514e-75c5-498e-bb37-a7fe962d4164
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
```

Hive query returns the **top five teams for the total scored points** in the table nba_table. Its groups with **PLAYER1_TEAM_NICKNAME**, and computes the total points scored by the team **SUM(POINTS_SCORED).** The list is ordered by the variable **total_points** in **descending order**. The last part is the **LIMIT 5**, which produces the five teams with the highest scores, revealing the best-performing teams in total points scored.

```
Ended Job = job_1734161408917_0007
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 8.34 sec   HDFS Read: 81597389 HDFS Write: 1073 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 4.87 sec   HDFS Read: 6820 HDFS Write: 219 SUCCESS
Total MapReduce CPU Time Spent: 13 seconds 210 msec
OK
Kings   8323.0
Bucks   8268.0
Mavericks       8211.0
Lakers  8068.0
Raptors 8005.0
Time taken: 51.939 seconds, Fetched: 5 row(s)
hive>
```

2. The Average Points Scored in Each Quarter.

```
SELECT PERIOD, SUM(POINTS_SCORED) / COUNT(DISTINCT GAME_ID) AS
nba_play_by_play
FROM nba_table
WHERE POINTS_SCORED IS NOT NULL AND POINTS_SCORED >= 0
GROUP BY PERIOD
ORDER BY PERIOD;
```

```
hive> SELECT PERIOD, SUM(POINTS_SCORED) / COUNT(DISTINCT GAME_ID) AS nba_play_by_play
    > FROM nba_table
    > WHERE POINTS_SCORED IS NOT NULL AND POINTS_SCORED >= 0
    > GROUP BY PERIOD
    > ORDER BY PERIOD;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using
execution engine (i.e. spark, tez) or using Hive 1.X releases.
Query ID = root_20241214083603_a8839afd-962b-402d-ad3e-e3be3b76d2c6
Total jobs = 2
Launching Job 1 out of 2
```

The Hive query obtains '**the points scored in each quarter (PERIOD) on average'** in all the games in the table nba_table. To **avoid invalid scores**, it starts from filtering the rows where **POINTS_SCORED** is **NULL or negative**. The output contains **average points** for that **quarter by taking total points scored** in that quarter and dividing them by the **number of games (GAME_ID)** held in that quarter. This query enabled the information to be displayed in a more organized and comprehensible manner by **quarter (PERIOD) and in ascending order** of the quarter.

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 13.89 sec   HDFS Read: 81599980 HDFS Write: 285 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 4.95 sec   HDFS Read: 5606 HDFS Write: 313 SUCCESS
Total MapReduce CPU Time Spent: 18 seconds 840 msec
OK
1       80.33564493758668
2       78.68055555555556
3       76.15117891816921
4       76.56527777777778
5       5.377104377104377
6       0.9613259668508287
7       0.6710526315789473
Time taken: 60.373 seconds, Fetched: 7 row(s)
hive>
```

3. Apache Spark

Google Colab, a cloud-based platform that supports **PySpark** for big data processing, was used to create this **Spark analysis task.**

A **Spark session** is started, and a notebook has been created. The dataset is then loaded.

```
from pyspark.sql import SparkSession

# Initialize the spark session
spark = SparkSession.builder.appName("NBA-Analysis").getOrCreate()
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
    v3.5.3
Master
    local[*]
AppName
    NBA-Analysis

```
# load the data
data = spark.read.csv("/content/drive/MyDrive/BDP-Spark/NBA-data.csv", header=True, inferSchema=True)
data.show()
```

```
+-------+--------+--------+--------------------+-------------------+------+----------+----------------+-------------------
|EVENTID|EVENTNUM| GAME_ID|     HOMEDESCRIPTION|       PCTIMESTRING|PERIOD|PLAYER1_ID|    PLAYER1_NAME|PLAYER1_TEAM_ABBREV
+-------+--------+--------+--------------------+-------------------+------+----------+----------------+-------------------
|      0|       0|20000001|                NULL|2024-12-14 12:00:00|     1|         0|            NULL|
|      1|       1|20000001|Jump Ball Camby v...|2024-12-14 12:00:00|     1|       948|    Marcus Camby|
|      2|       2|20000001|MISS Sprewell 6' ...|2024-12-14 11:41:00|     1|        84|Latrell Sprewell|
|      3|       3|20000001|                NULL|2024-12-14 11:40:00|     1|1610612755|            NULL|
|      4|       4|20000001|Camby S.FOUL (P1.T1)|2024-12-14 11:29:00|     1|       948|    Marcus Camby|
|      5|       5|20000001|                NULL|2024-12-14 11:29:00|     1|       689|    Theo Ratliff|
|      6|       6|20000001|                NULL|2024-12-14 11:29:00|     1|       689|    Theo Ratliff|
|      7|       7|20000001|Ward REBOUND (Off...|2024-12-14 11:28:00|     1|       369|    Charlie Ward|
|      8|       8|20000001|                NULL|2024-12-14 11:18:00|     1|       689|    Theo Ratliff|
|      9|       9|20000001|Camby Free Throw ...|2024-12-14 11:18:00|     1|       948|    Marcus Camby|
|     10|      10|20000001|Camby Free Throw ...|2024-12-14 11:18:00|     1|       948|    Marcus Camby|
|     11|      11|20000001|                NULL|2024-12-14 11:08:00|     1|       947|   Allen Iverson|
|     12|      12|20000001|Camby REBOUND (Of...|2024-12-14 11:06:00|     1|       948|    Marcus Camby|
|     13|      13|20000001|MISS Houston 15' ...|2024-12-14 10:53:00|     1|       275|   Allan Houston|
```

There are invalid values in the ‘SCORE’ column, such as ‘02-Feb’ and ‘Apr-6’. Create a function to clean and standardize these values by converting month names to their numerical equivalents as part of the preprocessing step.

```
# function for clean the score
def clean_score(score):
  # handle null score
  if score is None:
    return None
  if isinstance(score, str):
    month_mapping = {'Jan': '1', 'Feb': '2', 'Mar': '3', 'Apr': '4',
                     'May': '5', 'Jun': '6', 'Jul': '7', 'Aug': '8',
                     'Sep': '9', 'Oct': '10', 'Nov': '11', 'Dec': '12'}
    parts = score.split('-')
    if parts[0] in month_mapping:
      return f"{month_mapping[parts[0]]}-{parts[1]}"
    elif parts[1] in month_mapping:
      return f"{parts[0]}-{month_mapping[parts[1]]}"

    return score if '-' in score else None

  return None
```

The Spark code cleans the SCORE column, excluding null values from CLEANED_SCORE, and generates a PREVIOUS_SCORE column using a window partitioned by GAME_ID and ordered by EVENTID to capture the score from the prior event for sequential analysis.

```
# Apply cleaning function to the DataFrame
data = data.withColumn("CLEANED_SCORE", F.udf(clean_score,
                                               StringType())(data['SCORE']))

# Filter out rows where CLEANED_SCORE is null
cleaned_data = data.filter(data['CLEANED_SCORE'].isNotNull())

# Create a window specification to partition by GAME_ID and order by EVENTID
window_spec = Window.partitionBy("GAME_ID").orderBy("EVENTID")

# Initialize the previous score by using lag() function to get previous CLEANED_SCORE value
cleaned_data = cleaned_data.withColumn("PREVIOUS_SCORE",
                                        F.lag("CLEANED_SCORE").over(window_spec))
```

The **calculate_play_score** function is defined to compare the current and previous cleaned scores to determine the play score. It adds a new column, **PLAY_SCORE**, to the DataFrame and uses a UDF to determine the score difference or sum depending on whether a prior score exists.

```
# Function to calculate play score using UDF
def calculate_play_score(df):
  # inner function for calculate score
  def calculate_score(cleaned_score, previous_score):
    # If score is None, play score is 0
    if cleaned_score is None:
      return 0
    # First valid score, return sum of current score
    if previous_score is None:
      team1, team2 = map(int, cleaned_score.split('-'))
      return team1 + team2
    else:
      # Calculate difference between previous and current score
      previous_team1, previous_team2 = map(int, previous_score.split('-')) if previous_score else (0, 0)
      current_team1, current_team2 = map(int, cleaned_score.split('-'))
      return (current_team1 - previous_team1) + (current_team2 - previous_team2)

  # Create a UDF to apply the score calculation function row-wise
  calculate_score_udf = F.udf(calculate_score, IntegerType())

  return df.withColumn("PLAY_SCORE", calculate_score_udf(df['CLEANED_SCORE'], df['PREVIOUS_SCORE']))

# Apply the play score calculation
cleaned_data = calculate_play_score(cleaned_data)
```

1. % of Players (Scored 40 Points or More in A Single Match).

Computes the total number of players, counts the unique players who meet the condition, computes the percentage of such players, filters those with scores of 40 or higher, and groups the data by **PLAYER1_NAME** and **GAME_ID** to determine the total points per player.

```python
# Group by PLAYER1_NAME and GAME_ID and calculate the total points
player_score = cleaned_data.groupBy("PLAYER1_NAME" , "GAME_ID").agg(
    F.sum("PLAY_SCORE").alias("TOTAL_POINTS")
)

# filter player score 40 or more
filter_players = player_score.filter(player_score.TOTAL_POINTS >= 40)

# number of players
no_of_players = data.filter(col("PLAYER1_NAME").isNotNull()).select("PLAYER1_NAME").distinct().count()

# filter unique players who scored 40 or more points
filter_players_count = filter_players.filter(col("PLAYER1_NAME").isNotNull())\
.select("PLAYER1_NAME").distinct().count()

# Percentage of players who scored 40 or more points
player_percentage = (filter_players_count / no_of_players) * 100

print(f"The Percentage of players who scored 40 or more points in a single game: {player_percentage:.2f}%")

The Percentage of players who scored 40 or more points in a single game: 6.36%
```

- There are **6.36%** players are scored **40 or more** in a single game.

2. The Total Number of Matches Lost by Each Team.

Initially, computes the total **PLAY_SCORE** for each team by grouping data by **GAME_ID** and **PLAYER1_TEAM_NICKNAME.** It determines winners and losers by ranking teams in each game according to their overall score using a **window** function. The **Loser** teams are the only ones included in the filtered results. The code then sorts the teams in descending order of losses, counts the number of losses for each team, and groups the code by **PLAYER1_TEAM_NICKNAME.**

```python
# Group by GAME_ID and PLAYER1_TEAM_NICKNAME
# after calculate the sum of PLAY_SCORE for each team
game_scores = cleaned_data.groupBy("GAME_ID", "PLAYER1_TEAM_NICKNAME").agg(
    F.sum("PLAY_SCORE").alias("TOTAL_SCORE")
)

# window function to rank the teams within each game by their total score
window_spec = Window.partitionBy("GAME_ID").orderBy(F.col("TOTAL_SCORE").desc())

# Add a column to rank teams based on their total score within each game
game_scores = game_scores.withColumn("RANK", F.row_number().over(window_spec))

# identify the winner and loser
game_results = game_scores.withColumn(
    "RESULT",
    F.when(F.col("RANK") == 1, "Winner").otherwise("Loser")
)

# filter using PLAYER1_TEAM_NICKNAME
game_results = game_results.filter((F.col("PLAYER1_TEAM_NICKNAME").isNotNull()))
game_results.show(truncate=False)
```

```python
# Filter the game results to include only the "Loser" teams
losses = game_results.filter(
    (F.col("RESULT") == "Loser") & (F.col("PLAYER1_TEAM_NICKNAME").isNotNull())
)
```

```python
# Group by PLAYER1_TEAM_NICKNAME and count the number of losses for each team
team_losses = losses.groupBy("PLAYER1_TEAM_NICKNAME").agg(
    F.count("RESULT").alias("NUMBER_OF_LOSSES")
)
```

```python
# Sort the teams by the number of losses in descending order
team_losses_sorted = team_losses.orderBy(F.col("NUMBER_OF_LOSSES").desc())
team_losses_sorted.show(truncate=False)
```

```
+---------------------+----------------+
|PLAYER1_TEAM_NICKNAME|NUMBER_OF_LOSSES|
+---------------------+----------------+
|Bulls                |67              |
|Warriors             |65              |
|Wizards              |63              |
|Grizzlies            |59              |
|Hawks                |57              |
|Nets                 |56              |
|Cavaliers            |52              |
|Clippers             |51              |
```

- Team Bulls lost most matches (67) in this tournament.

Task 3 - Machine Learning model using Spark MLlib

The Spark session is already initialized, and the data has been loaded in Google Colab.

To keep every record from the **cleaned_data** dataset, do a left join. Save the result in a **new DataFrame** for additional processing after replacing null values in the **PLAY_SCORE** column with 0.

```
# Perform a left join to retain all records from cleaned data
merged_data = data.join(
    cleaned_data.select("GAME_ID", "EVENTID", "PLAY_SCORE"),
    on=["GAME_ID", "EVENTID"],
    how="left"
)

# Fill null values in the PLAY_SCORE column with 0
new_data = merged_data.withColumn(
    "PLAY_SCORE",
    F.when(F.col("PLAY_SCORE").isNull(), 0).otherwise(F.col("PLAY_SCORE"))
)

# Show the resulting DataFrame
new_data.show()
```

Calculate the total points scored by each team in a single game after grouping the data by **GAME_ID** and **PLAYER1_TEAM_NICKNAME.** Teams in each game can be ranked according to their total points by using a **window function.** To decide which team **won** and which **lost**, assign ranks. This makes it possible to identify the winning and losing teams for each match.

```
# Determine total points scored by each team per game
team_scores = (
    new_data.groupBy("GAME_ID", "PLAYER1_TEAM_NICKNAME")
    .agg(F.sum("PLAY_SCORE").alias("TOTAL_POINTS"))
)

# Determine winners and losers
game_results = (
    team_scores.alias("a")
    .join(
        team_scores.alias("b"),
        (F.col("a.GAME_ID") == F.col("b.GAME_ID")) &
        (F.col("a.PLAYER1_TEAM_NICKNAME") != F.col("b.PLAYER1_TEAM_NICKNAME")),
    )
    .select(
        F.col("a.GAME_ID").alias("Game_ID"),
        F.col("a.PLAYER1_TEAM_NICKNAME").alias("Winner"),
        F.col("b.PLAYER1_TEAM_NICKNAME").alias("Loser"),
    )
    .filter(F.col("a.TOTAL_POINTS") > F.col("b.TOTAL_POINTS"))
)
```

Only players from the winning team should be included in the data. Next, group the filtered data by **PLAYER1_NAME**, and then add up the **PLAY_SCORE** column to determine the **total points scored by each player.** This shows how much each player on the winning team contributed overall.

```
# Filter players on the winning team and calculate total points per player
winning_players_total_score = (
    new_data.alias("players")
    .join(game_results.alias("results"),
          (F.col("players.GAME_ID") == F.col("results.Game_ID")) &
          (F.col("players.PLAYER1_TEAM_NICKNAME") == F.col("results.Winner")),
          "inner")
    .groupBy("players.GAME_ID", "players.PLAYER1_NAME", "players.PLAYER1_TEAM_NICKNAME", "results.Loser")
    .agg(F.sum("players.PLAY_SCORE").alias("Total_Points"))
    .select(
        F.col("GAME_ID").alias("Game_ID"),
        F.col("PLAYER1_NAME").alias("Player_Name"),
        F.col("PLAYER1_TEAM_NICKNAME").alias("Team"),
        F.col("Loser").alias("Opponent"),
        F.col("Total_Points").alias("Points_Scored")
    )
)

# Show the results
winning_players_total_score.show()
```

```
+--------+-----------------+---------+--------+-------------+
| Game_ID|      Player_Name|     Team|Opponent|Points_Scored|
+--------+-----------------+---------+--------+-------------+
|20000002|    Lamond Murray|Cavaliers|    Nets|           17|
|20000002|Clar. Weatherspoon|Cavaliers|    Nets|            8|
```

Preparing the data is the first step in the **Machine Learning** process.

```
model_data = winning_players_total_score.select(
    F.col("Player_Name").alias("Player"),
    F.col("Opponent"),
    F.col("Points_Scored").alias("Actual_Score")
)
```

To prepare the data for machine learning models, use **OneHotEncoder** to encode the string columns **Player** and **Opponent** into numerical format.

```
from pyspark.ml.feature import StringIndexer

player_indexer = StringIndexer(inputCol="Player", outputCol="Player_Index")
opponent_indexer = StringIndexer(inputCol="Opponent", outputCol="Opponent_Index")
```

```
indexed_data = player_indexer.fit(model_data).transform(model_data)
indexed_data = opponent_indexer.fit(indexed_data).transform(indexed_data)
```

```
from pyspark.ml.feature import OneHotEncoder

# OneHotEncoder for Player and Opponent
player_encoder = OneHotEncoder(inputCol="Player_Index", outputCol="Player_OneHot")
opponent_encoder = OneHotEncoder(inputCol="Opponent_Index", outputCol="Opponent_OneHot")

encoded_data = player_encoder.fit(indexed_data).transform(indexed_data)
encoded_data = opponent_encoder.fit(encoded_data).transform(encoded_data)
```

To get the data ready for machine learning modelling, use **VectorAssembler** to merge several feature columns into a single feature vector.

```
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(
    inputCols=["Player_OneHot", "Opponent_OneHot"],
    outputCol="features"
)

data_features = assembler.transform(encoded_data)
```

After that, Split the data into training and testing

```
train_data, test_data = data_features.randomSplit([0.8, 0.2], seed=42)
```

Define the models: **Linear Regression**, **Random Forest Regression**, **GBT Regression**, and **Decision Tree Regression**, to train and evaluate on the prepared data.

```
from pyspark.ml.regression import (
    LinearRegression,
    RandomForestRegressor,
    GBTRegressor,
    DecisionTreeRegressor
)
from pyspark.ml.evaluation import RegressionEvaluator

# build the models
models = {
    "Linear Regression": LinearRegression(featuresCol="features", labelCol="Actual_Score"),
    "Random Forest Regression": RandomForestRegressor(featuresCol="features", labelCol="Actual_Score"),
    "GBT Regression": GBTRegressor(featuresCol="features", labelCol="Actual_Score"),
    "Decision Tree Regression": DecisionTreeRegressor(featuresCol="features", labelCol="Actual_Score"),
}
```

Build the model that was selected, use the training data to train it, and then use the testing data to make predictions. Lastly, use the **R2** value and **Root Mean Squared Error (RMSE)** to evaluate the accuracy and fit of the model.

```
# Train and Evaluate every model
results = []
evaluator = RegressionEvaluator(labelCol="Actual_Score", predictionCol="prediction", metricName="rmse")

for name, model in models.items():
    # Train the model
    trained_model = model.fit(train_data)
    # Predict on the test data
    predictions = trained_model.transform(test_data)
    # Evaluate RMSE
    rmse = evaluator.evaluate(predictions)
    # Evaluate R2
    r2 = evaluator.setMetricName("r2").evaluate(predictions)
    # Append results
    results.append((name, rmse, r2, predictions.select("Player", "Opponent", "Actual_Score", "prediction")))
```

```python
print("Model Comparison Results:")
for name, rmse, r2, predictions_df in results:
    print(f"\nModel: {name}")
    print(f"  RMSE: {rmse:.3f}")
    print(f"  R2: {r2:.3f}")
```

```
Model Comparison Results:

Model: Linear Regression
  RMSE: 6.687
  R2: 0.483

Model: Random Forest Regression
  RMSE: 0.108
  R2: 0.108

Model: GBT Regression
  RMSE: 0.260
  R2: 0.260

Model: Decision Tree Regression
  RMSE: 0.077
  R2: 0.077
```

Out of the four models used in this evaluation, the Decision Tree Regression model has the lowest RMSE and is therefore the most accurate. While Random Forest and GBT regressions yield results in the middle, Linear Regression performs the worst.0

Use a **parameter grid** to perform **Hyperparameter Tuning** for the **Linear Regression** model, and then use **5-fold cross-validation** to test various parameter combinations and optimize model performance.

```python
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# Define the Linear Regression model
linear_model = LinearRegression(featuresCol="features", labelCol="Actual_Score")

# Create a parameter grid for hyperparameter tuning
paramGrid = ParamGridBuilder().addGrid(linear_model.regParam,
 [0.01, 0.1, 0.5, 1.0]).addGrid(linear_model.elasticNetParam,
  [0.0, 0.5, 1.0]).build()

# Define the evaluator for RMSE
evaluator = RegressionEvaluator(labelCol="Actual_Score",
                                predictionCol="prediction",
                                metricName="rmse")

# Set up CrossValidator
crossval = CrossValidator(estimator=linear_model,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=5)
```

```python
# Train the model with cross-validation
cv_model = crossval.fit(train_data)

# Get the best model
best_model = cv_model.bestModel

# Evaluate the model on test data
predictions = best_model.transform(test_data)

# Calculate RMSE and R2 on test data
rmse = evaluator.evaluate(predictions)
r2_evaluator = RegressionEvaluator(labelCol="Actual_Score",
                                  predictionCol="prediction",
                                  metricName="r2")
r2 = r2_evaluator.evaluate(predictions)
```

```python
# Show predictions
print("Best Linear Regression Model Predictions (Top 5):")
predictions.select("Player", "Opponent", "Actual_Score", "prediction").show(5)

# Output the evaluation metrics
print(f"Best Linear Regression Model - RMSE: {rmse}, R2: {r2}")

# Print the best hyperparameters
print(f"Best regParam: {best_model._java_obj.getRegParam()}")
print(f"Best elasticNetParam: {best_model._java_obj.getElasticNetParam()}")
```

```
Best Linear Regression Model Predictions (Top 5):
+----------+---------+------------+-----------------+
|    Player| Opponent|Actual_Score|       prediction|
+----------+---------+------------+-----------------+
|A.C. Green|    Bulls|           0|5.373035715762285|
|A.C. Green|  Celtics|          12|5.629694488773146|
|A.C. Green|Grizzlies|           2|5.391726663948985|
|A.C. Green|     Jazz|           0|4.786912338401125|
|A.C. Green|  Raptors|          14|5.833808353523322|
+----------+---------+------------+-----------------+
only showing top 5 rows

Best Linear Regression Model - RMSE: 6.686678555332445, R2: 0.48263004170596113
Best regParam: 0.01
Best elasticNetParam: 1.0
```

Task 4 – Data Visualization