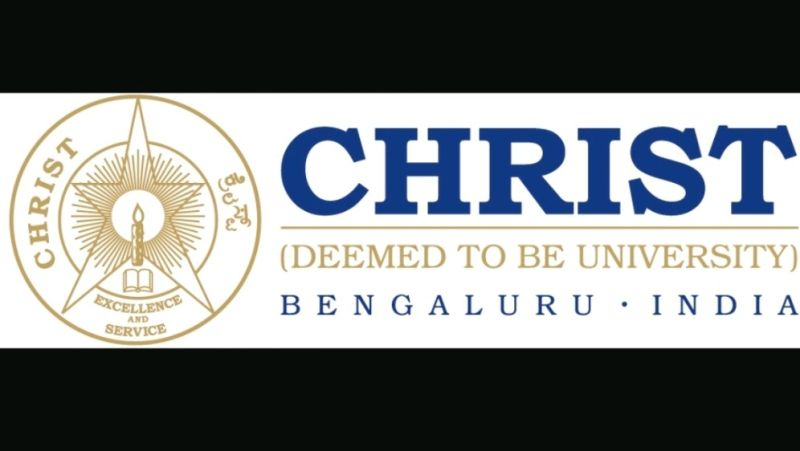
****

**US**

**(DEMONSTRATION OF DISTRIBUTED CACHE CONCEPTS)**

**SANJAY R**

**2348055**

**15th November 2024**

**MDS571**

**[Big Data Analytics](https://classroom.google.com/c/NzA2NzQxNDE2MTc1" \t "https://classroom.google.com/c/_self)**

**Department of Statistics and Data Science**

**CHRIST (Deemed to be University)**

**Department of Statistics and Data Science**

**Course:MDS571-[Big Data Analytics](https://classroom.google.com/c/NzA2NzQxNDE2MTc1" \t "https://classroom.google.com/c/_self) Lab :5**

**R.SANJAY 2348055**

*User*

*DEMONSTRATION OF DISTRIBUTED CACHE CONCEPTS*

* **INTRODUCTION:**

In today’s data-centric world, the healthcare industry generates a massive amount of information daily, from patient records to medical research data. However, this data is often fragmented and requires significant processing to make it useful for decision-making. One of the common challenges faced in this domain is linking clinical data with contextual information such as disease metadata. For instance, patient records may contain disease codes, but without proper mapping to descriptions, these records remain cryptic and difficult to interpret.

This project focuses on addressing this issue by developing a distributed data processing solution using Hadoop's MapReduce framework. The goal is to integrate patient data with disease metadata, thereby enhancing the usability and interpretability of the data. The implementation utilizes Hadoop’s DistributedCache to efficiently handle metadata distribution across the cluster nodes, ensuring scalable and high-performance processing. By automating the mapping of disease codes to their respective descriptions, this project provides a robust solution for enriching healthcare datasets, paving the way for more informed analytics and decision-making in the healthcare sector.

This endeavor highlights the practical application of big data technologies in solving complex real-world problems, demonstrating how distributed computing can be leveraged to process large datasets seamlessly and efficiently.

* **PROBLEM DESCRIPTION:**

In healthcare analytics, one of the key challenges is integrating disparate datasets to derive meaningful insights. Patient records, which contain critical information such as personal details, age, gender, and diagnosed disease codes, are often stored separately from disease metadata that provides contextual information about these codes. This separation creates significant hurdles in understanding and analyzing the data effectively. For example, a dataset with a column for "Disease\_Code" can provide little insight on its own without a corresponding description or category to explain what the code represents.

This problem becomes more complex when dealing with large-scale data in distributed systems, where multiple nodes process different parts of the data. Ensuring that every processing node has access to the disease metadata file is critical for seamless integration. Manual methods of joining such datasets are not only time-consuming but also prone to human error, especially when datasets are large or updated frequently.

To address these challenges, we propose a MapReduce solution that performs a distributed join between patient data and disease metadata. This approach ensures scalability, accuracy, and efficiency in enriching patient records with meaningful disease information. The goal is to produce an output dataset that combines patient details with descriptive and categorical disease information, thereby enabling better healthcare analytics and decision-making.

* **DATASET DESCRIPTION:**

The program utilizes two key datasets: patient\_data and disease\_metadata, which together provide a comprehensive view of a cricketer's profile.

* **patient\_data:**

The Patient Data dataset contains detailed information about individual patients and their associated disease codes. Below are the column names and their descriptions:

* **Patient\_ID:** A unique identifier for each patient.
* **Name:** The name of the patient.
* **Age:** The age of the patient.
* **Gender:** The gender of the patient (e.g., Male, Female).
* **Disease\_Code:** A code that represents a specific disease diagnosed for the patient.
* **disease\_metadata:**

The Disease Metadata dataset provides additional information about diseases, enriching the disease codes with contextual data. Below are the column names and their descriptions:

* **Disease\_Code:** A unique code identifying each disease (matches with the codes in the Patient Data dataset).
* **Disease\_Description:** A brief description of the disease corresponding to the code.
* **Disease\_Category:** A category or classification to which the disease belongs, such as chronic, infectious, etc.

Together, these datasets enable the generation of comprehensive patient records by linking descriptive and categorical disease information to individual patients.

Both datasets are structured in a tab-separated format, with each record corresponding to a specific player. The player\_id field acts as the common key between the two datasets, which is used to merge the data. For effective integration, the data in these datasets are preprocessed to ensure consistency in formatting and are aligned for easy use during the MapReduce process.

These datasets reflect real-world cricket statistics, offering a solid basis for analyzing player performance. The design and structure of the datasets ensure they are both relevant and applicable for joining operations, enabling insightful analysis of player performance when combined.

* **CODE:**

*import org.apache.hadoop.conf.Configuration;*

* **PROGRAM DESCRIPTION:**

The Cricket Dataset Join Program utilizes Hadoop's MapReduce framework to efficiently integrate two large datasets containing cricket-related information. This program is specifically designed to handle the challenges of merging datasets in a distributed environment, where the volume of data can be substantial. By leveraging the MapReduce paradigm, the program ensures scalability, fault tolerance, and high performance during the data processing stage. The task involves joining two datasets—player\_info and performance\_info—on a common key, player\_id, to produce a unified output that consolidates player demographics and performance statistics.

The program's architecture is divided into three primary components—Mapper, Reducer, and Driver. Each of these components plays a distinct role in transforming and integrating the data, from reading and categorizing it to joining and outputting the final results. Below is a detailed explanation of how these components collaborate to achieve the program's objectives.

* **MAPPER CLASS:**

The Mapper class serves as the entry point for processing the input datasets. It reads the player\_info and performance\_info files line by line, where each record is tab-separated. The key task of the Mapper is to emit player\_id as the key and the entire record as the value. This enables the MapReduce framework to group all records associated with a particular player\_id during the shuffle and sort phase. Unlike traditional approaches where data may be pre-categorized, the Mapper is designed to treat input records uniformly, deferring dataset differentiation to the Reducer. This approach simplifies the Mapper logic and ensures flexibility when dealing with datasets of varying structures. By producing key-value pairs for each record, the Mapper sets the foundation for efficient joining of datasets.

* **REDUCER CLASS:**

The Reducer class takes over after the shuffle and sort phase, where all records with the same player\_id are grouped together. It is responsible for merging the records from the two datasets based on the shared player\_id. To achieve this, the Reducer identifies the origin of each record (whether from player\_info or performance\_info) by analyzing the number of fields in the value. Once identified, it combines the information into a unified record, ensuring that all details about a player are consolidated into a single output line. The Reducer outputs these merged records in a tab-separated format, ready for further analysis. By handling dataset differentiation and integration, the Reducer ensures that the final output is both comprehensive and logically ordered.

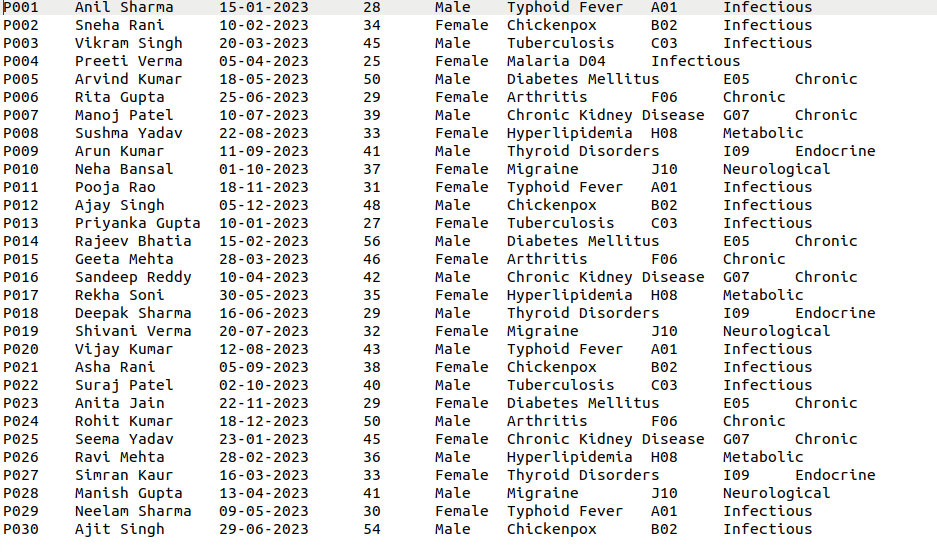
* **DRIVER CLASS:**

The Driver class acts as the controller and coordinator for the entire MapReduce program. It is responsible for configuring the job, specifying the Mapper and Reducer classes, and defining the input and output formats. The Driver also sets the paths for the input datasets and output directory, ensuring seamless data flow. Additionally, it includes error handling mechanisms to validate input arguments and manage exceptions during execution. Once the job is configured, the Driver submits it to the Hadoop cluster for execution and monitors its progress until completion. The Driver's role is crucial in orchestrating the interaction between the Mapper and Reducer, ensuring that the program runs smoothly from start to finish.

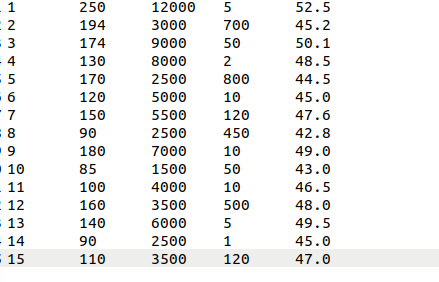
* **PROJECT SETUP AND EXECUTION:**
* **DATASET PREPARATION:**

The Cricket Dataset Join Program relies on two primary datasets: player\_info and performance\_info.

* **player\_info Dataset:** This dataset includes player demographic information such as player\_id, name, country, and role. Each record provides critical details necessary for identifying and categorizing players.

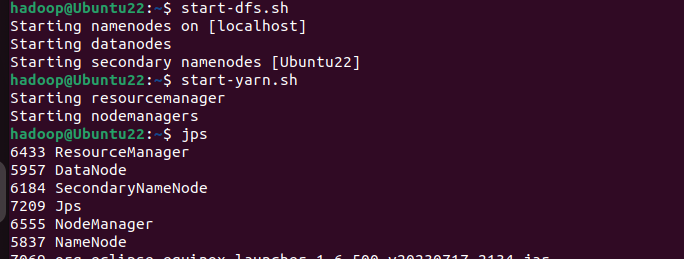


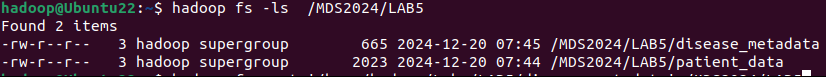
* **performance\_info Dataset:** This dataset captures players' performance statistics, including player\_id, matches, runs, wickets, and averages. These metrics are essential for analyzing player performance and contributions in cricket.

****

Both datasets were carefully formatted to ensure consistency, with player\_id acting as the common key to join the two. Each record is stored in a tab-separated format, which is optimal for processing with Hadoop's TextInputFormat.

To facilitate processing with the Hadoop MapReduce framework, the datasets were placed within the Hadoop directory structure at /MDS2024/LAB4. This setup ensured that the files were easily accessible for the program, allowing seamless integration and efficient execution of the MapReduce process.

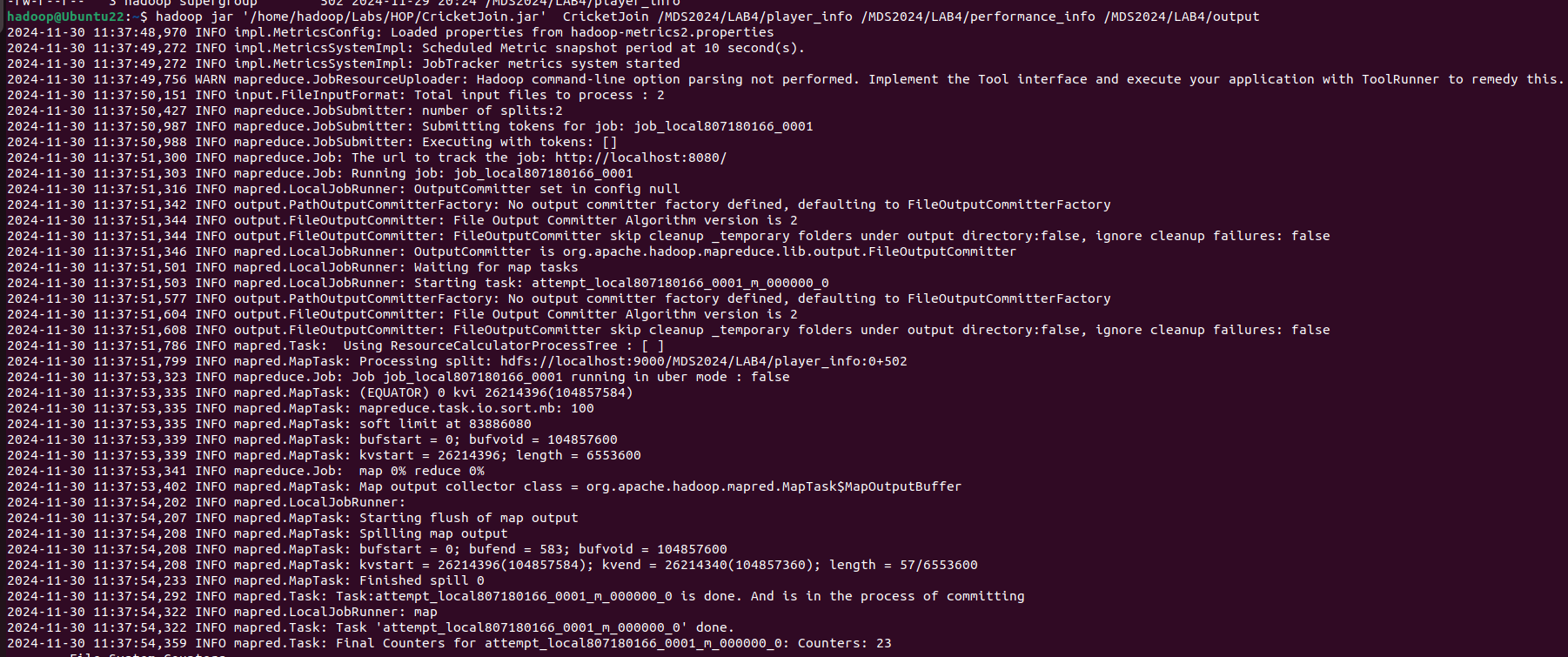




* **ECLIPSE SETUP:**

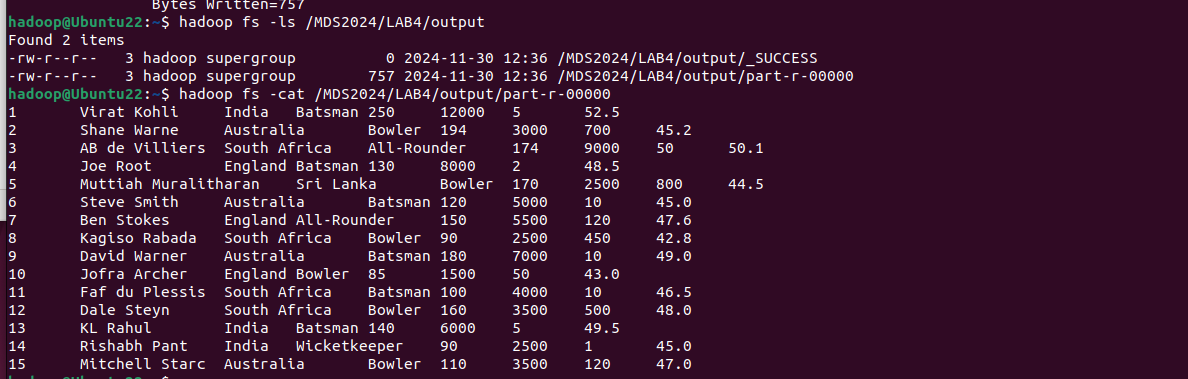
To develop the Cricket Join Program, the Eclipse IDE was utilized as the primary platform for Java development. Known for its reliability and comprehensive features, Eclipse provided a robust environment for coding, testing, and debugging the Hadoop MapReduce components. The development process involved the following steps:

* A new Java project named CricketJoin was created within Eclipse. This project served as the foundational framework for implementing the MapReduce logic required for joining cricket datasets.
* **Main Class and MapReduce Components:** Within the CricketJoin project, the main class was implemented to orchestrate the Mapper, Reducer, and Driver components. Each component played a specific role in the program:
* The CricketMapper class was designed to process input records from both datasets (player\_info and performance\_info).It extracted the player\_id field from each line and emitted it as the key, along with the full record as the value.This approach enabled the Reducer to distinguish between the datasets based on their structure and join the relevant data efficiently.
  + - The CricketReducer class received grouped records based on the common key player\_id.It identified the source dataset of each record using the number of fields and merged the corresponding player demographic information with performance statistics.The Reducer then wrote the unified records to the output, ensuring all player details and performance metrics were consolidated into a single, structured line.
* After the code was developed and thoroughly tested, the necessary Hadoop library JAR files were added to the project to ensure Hadoop-specific functionalities during execution. The project was then exported as a JAR file named CricketJoin.jar, making it ready for deployment within the Hadoop environment.
* **EXECUTION PROCESS:**
* The next step involved executing the program within the Hadoop environment, using the following command:

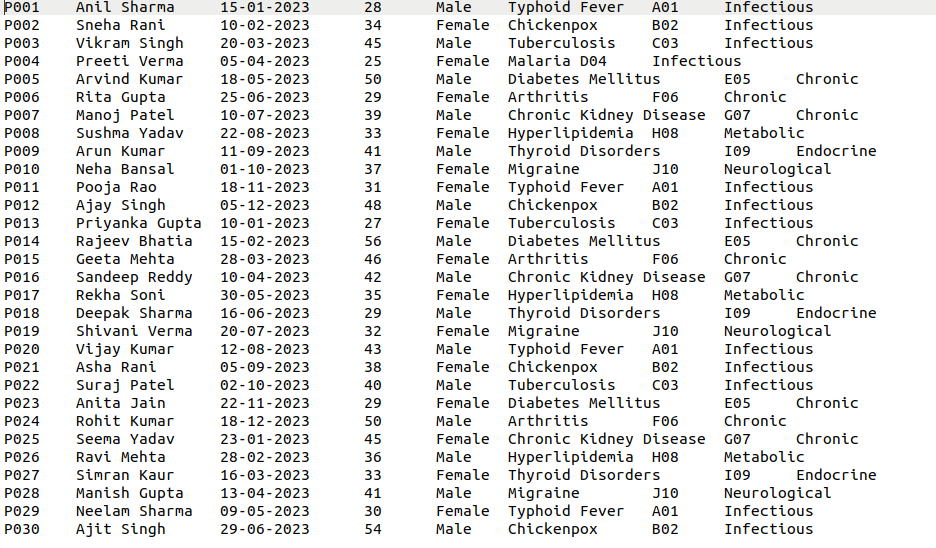
**$ hadoop jar '/home/hadoop/Labs/HOP/CricketJoin.jar' CricketJoin /MDS2024/LAB5/player\_info /MDS2024/LAB5/performance\_info /MDS2024/LAB5/output**

**~$ hadoop fs -ls /MDS2024/LAB5/output**

**~$ hadoop fs -ls /MDS2024/LAB5/output/part-r-00000**



* **OUTPUT:**



* **CONCLUSION:**

This activity demonstrates the power of distributed computing in solving real-world data integration challenges. By utilizing Hadoop's MapReduce framework and the DistributedCache feature, the program processes patient data at scale, mapping disease codes to descriptions seamlessly. The integration of metadata into patient records not only enriches the dataset but also lays the groundwork for further healthcare analytics. This implementation exemplifies how modern tools and frameworks can transform raw data into actionable insights, addressing key challenges in the healthcare domain.