**3135194 ITNPBD7 PROJECT REPORT**

**Part 1**

**Below are some Hadoop commands needed to prepare HDFS space, move data to/fro, run code, clean-up and carrying-out other task**

**Five commands used and the explanations**

1. **hdfs dfs -mkdir /user/dcu00002/Project:**

I used this sub-command to create a directory within the Hadoop Distributed File System (HDFS). before running a Hadoop job, I needed to ensure that the necessary directories exist in HDFS to enable storage of input data within a specific location.

**2) hdfs dfs -ls**

I used this sub-command to lists the contents of a directory in HDFS. It is useful for verifying that files and directories have been successfully created or copied to HDFS before running a Hadoop job.

**3) hdfs dfs -chmod**

I used this command to change the permissions of files or directories in HDFS. It's necessary for controlling access to data and ensuring data security within HDFS.

1. **hdfs dfs -cat /user/dcu00002/Project/output/part-00000**:

I used this sub-command to display the contents of my file stored in HDFS. It is useful for inspecting the output of my Hadoop job, directly in the terminal. In general , It allows quick check of the results of the MapReduce in the terminal without needing to download the files back to the local system.

1. **hdfs dfs -rm -r /user/dcu00002/Project/output:**

It permanently removes or deletes the output directory and its contents from HDFS

**PART 2**

**DESIGNS**

**\* MAPPER DESIGN**

**INPUT :** for the input, each text lines from the rating.txt file, each containing a movie's record (Reviewer\_ID, movie\_title, genre, year, rating).

**ALGORITHM:** the code will breakdown each input record, extract relevant fields, split genres if multiple, filter by years of interest.

**OUTPUT :** will output  **year**,the **movie titles**, **ratings**, and **count**.

**\* COMBINER DESIGN**

**INPUT :** Sort key-value pairs from the mapper, representing movie ratings, each containing year, title, rating, and count.

**ALGORITHM:** iterates through the input data, accumulating ratings and counts for each movie, and getting the average rating for each movie .

**OUTPUT :** print the year, title, average rating, and summed count for each movie to the standard output.

**\* REDUCER DESIGN**

**INPUT :** Aggregated key-value pairs from combiners, data containing the year, title, rating, and count of votes for movies

**ALGORITHM:** aggregate ratings and counts for each movie per year, then identify and print movies with the highest average rating annually, while considering a minimum vote threshold

.

**OUTPUT :** output the year, movie titles, and the highest average rating for each year.

**PART 3**

**PART 4**

Thinking about what would be involved in completing this task on a distributed computation cluster such as Condor instead of using Hadoop

Condor, a distributed system shows several differences compared to Hadoop, particularly when dealing with large datasets. While Condor offers excellent flexibility and efficiency for high-throughput computing tasks across a wide range of job types, the Hadoop ecosystem provides a more integrated solution for handling petabyte-scale data processing tasks. However, specific use cases, particularly those emphasizing compute over data transfer, could still benefit from Condor’s high-throughput capabilities.

**Some of the Pros of Using Condor:**

\* **High Throughput**: Condor is designed for high-throughput computing (HTC), making it well-suited for a large number of small jobs.

\* **Resource Optimization**: Condor utilizes available computing resources by distributing tasks to desktop computers and servers, potentially offering higher resource availability.

\* **Flexible Scheduling**: It offers advanced job queueing mechanisms, and priority scheduling to manage workloads effectively.

\* **Variety of Workloads and Checkpointing**: Supports a wide range of job types, not limited to MapReduce jobs. Also, it provides support for checkpointing, allowing long-running jobs to be paused and resumed.

**Some of the Cons of Using Condor:**

**\* Complexity for Data-Intensive Tasks**: For data-intensive tasks, especially those requiring frequent access to a large dataset (like petabyte-scale), managing data locality and efficient data transfer becomes more complex compared to Hadoops HDFS.

**\* Lack of Integrated Data Storage:** Unlike Hadoop, which has HDFS for distributed storage closely integrated with its processing model, Condor does not have a built-in distributed file system, potentially complicating data management and access.

**\* Setup and Maintenance:** Setting up a Condor pool might require more configuration and maintenance effort, especially to optimize for data-intensive tasks, compared to Hadoop clusters which are widely used for such tasks and have more turnkey solutions available.

**Between Condor and Hadoop the best framework to use to analyse the data set for this assignment :** Hadoop is the preferred choice because the dataset is large and requires distributed data processing.

In distributed Data Processing, The dataset provided in the assignment is a large dataset. Processing such a large dataset efficiently would require distributed data processing capabilities, which Hadoop provides through its MapReduce framework. Hadoop's ability to distribute the workload across multiple nodes in a cluster ensures that the analysis can be performed in a scalable and efficient manner.

In built-in Mechanisms for Data Handling, Hadoop comes with built-in support for distributed storage (HDFS) and distributed processing (MapReduce), making it well-suited for handling large volumes of data that may not fit into memory on a single machine. This is essential for processing big data efficiently.

Considering MapReduce Programming Model, The MapReduce paradigm provided by Hadoop allows for the parallel processing of data across a distributed cluster of machines. This is particularly suitable for tasks like finding the movies with the highest average ratings for each year, where the computation can be parallelized across multiple nodes.

In terms of scalability and Fault Tolerance, Hadoop is designed to be highly scalable and fault-tolerant. It can scale horizontally by adding more nodes to the cluster as the dataset grows, and it automatically handles node failures, ensuring that the analysis can continue uninterrupted even in the presence of hardware failures.