Map/reduce Implementation

CSE 5331: Project 3

Krishna Prabhakar Chinya

Table of Contents

[Project statement 2](#_Toc468907583)

[Implementation 2](#_Toc468907584)

[Reducer: 2](#_Toc468907585)

[Overall Status 3](#_Toc468907586)

[Division of Labour 3](#_Toc468907587)

[Logical Errors 4](#_Toc468907588)

[Report 4](#_Toc468907589)

# Projectstatement

We were given a temperature dataset that had temperatures of the past 5 years of Texas. We had to design and develop a map program and a reduce program to sort the data set on temperature.

To understand and appreciate the power of this paradigm and the ease of scaling using this paradigm, we had to run the same data set:

i). On multiple mappers and using equal temperature range

ii). By sampling the data set to determine the temperature range better so that the number of records in each range is not skewed.

iii). And measure response time and number of records in each range for the above two approaches (and other measures as applicable) and compare and analyse them.

The input data is typically partitioned (sharded) into 64MB splits by default.

# Implementation

Mapper:

Mapper takes the sharded input and we do mapping of key and value as per our needs. Here we have chosen Key as Temperature and Value for the date column. In Mapper we are filtering the temperature as per the needs. Since the given input file in not .csv we have to use the regular expression in order to divide the data properly.

Input:

690190 13910 20060201\_0 51.75 33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 000000

690190 13910 20060201\_1 54.74 33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 000000

690190 13910 20060201\_2 50.59 33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 000000

690190 13910 20060201\_3 51.67 33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 000000

Output: After Shuffle and sort

50.59 20060201\_0

51.67 20060201\_0

51.75 20060201\_0

54.74 20060201\_0

# Reducer:

Reducer takes the input from the mapper and combines the result for of each key as the user has written the code. Input for the reducer is a key and a collection of values. In our example reducer gets the input from the mapper/combiner and input is the key which is Temperature. Value is the Date.

Input*:*

50.59 20060201\_0

51.67 20060201\_0

51.75 20060201\_0

54.74 20060201\_0

Output*:*

*F*ile1

50.59 20060201\_0

51.67 20060201\_0

*File2*

51.75 20060201\_0

54.74 20060201\_0

# Overall Status

We have implemented the map reduce for the intended problem and we have succeeded in getting the result on Single node cluster.

In this project we have implemented two types of practitioner, one is customer practitioner and another we have used the built practitioner (Total Order Practitioner).

# Division of Labour

This project is implemented by a team of two members. The both team members have equally contributed for the successful completion of the project. The distribution of project work is as following

Krishna Prabhakar Chinya:

Krishna has implemented the Map/Reduce for the given data set, and he has also installed the single node cluster setup of Hadoop in his system.

We did a great team work by reviewing both of our work each other, and we had great brainstorming sessions to understand the requirements of Map/reduce and increasing the performance of the map/reduce

Krishna also did the testing of the our project like checking the whether the our logic for practitioner is work as need by checking the boundary condition for the different set of input and output

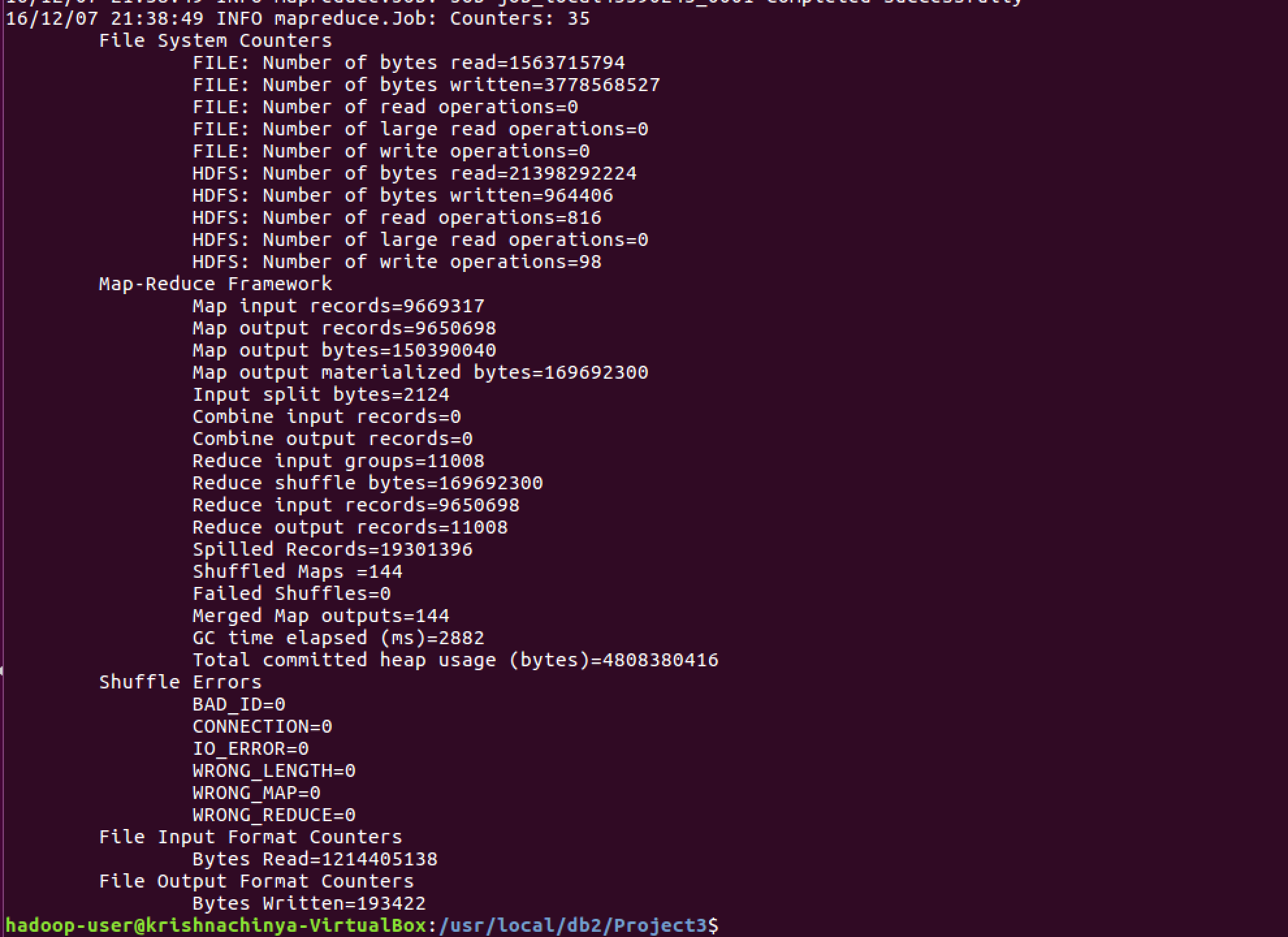
# Logical Errors

There were some logical errors that we have faced during installation of the Hadoop. Nonetheless, we had minimal error in implementing the map/reduce.

# Report

Below the output and run times for the custom and Total order partitioner.

**Custom Partitioner**



Stats from Custom Partitioner

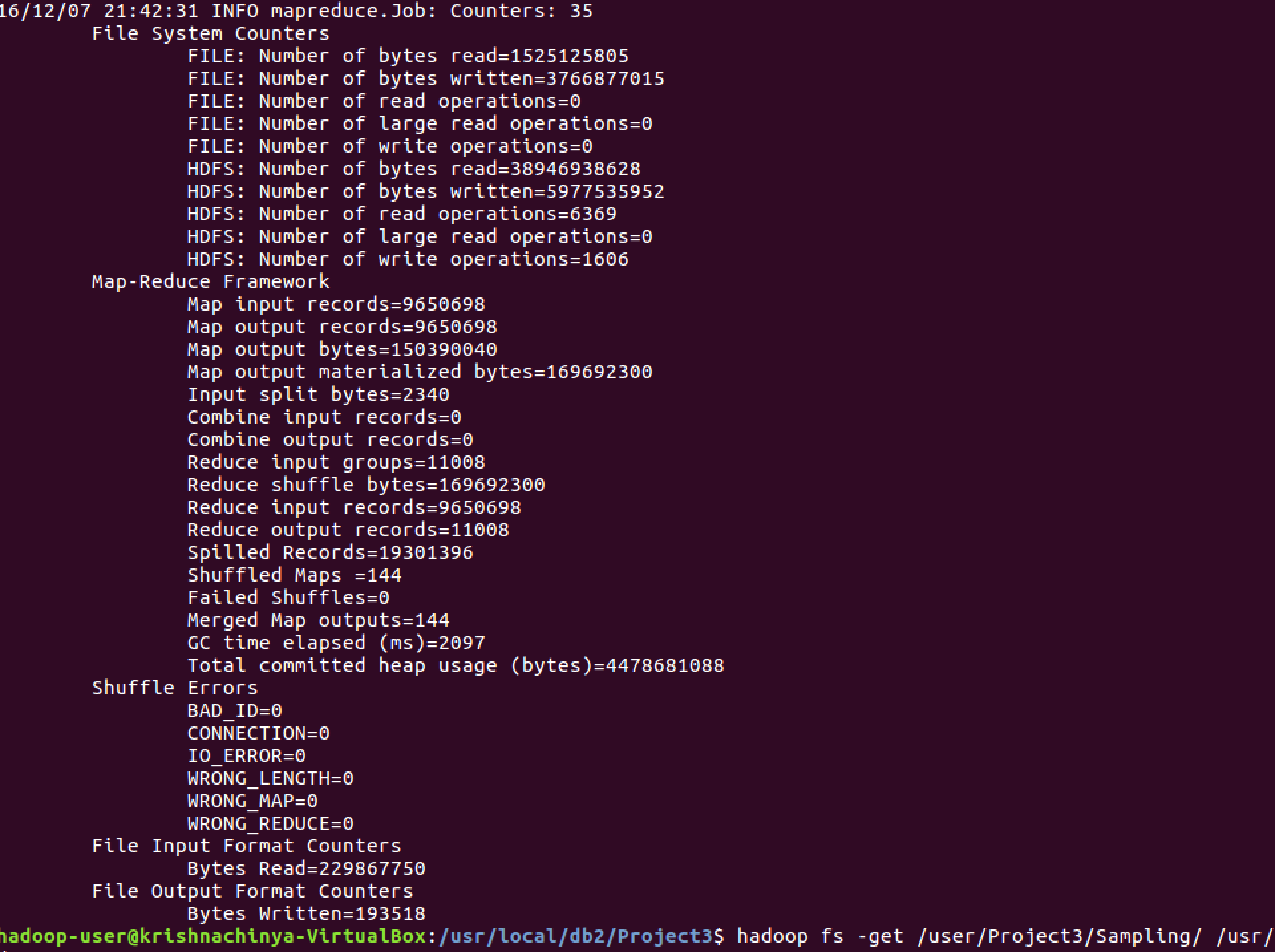
Amount of Time: 1:33:37

Number of Reducer: 8

Data Distribution

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Reducer1 | Reducer2 | Reducer3 | Reducer4 | Reducer5 | Reducer6 | Reducer7 | Reducer8 |
| 1210 | 1750 | 1750 | 1750 | 1750 | 1750 | 736 | 312 |
| 11% | 15% | 15% | 15% | 15% | 15% | 6.8% | 2.8% |

**Total Order Partitioner**



Amount of Time: 2:24:30

Number of Reducer: 8

Data Distribution

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Reducer1 | Reducer2 | Reducer3 | Reducer4 | Reducer5 | Reducer6 | Reducer7 | Reducer8 |
| 4168 | 1058 | 751 | 580 | 475 | 447 | 520 | 3009 |
| 37.8% | 9.6% | 6.8% | 5.2% | 4.3% | 4% | 4.72% | 27.3% |

Here our custom partitioner worked better than the Total Order partitioner.

**REFERENCES**

1. Database Management System 3rd Edition book by Raghu Ramakrishnan and Johannes Gehrke.

2. https://wweb.uta.edu/faculty/sharmac/javadocs/index.html

3. <https://elearn.uta.edu/bbcswebdav/pid-4624294-dt-content-rid-39372361_2/courses/2162-DBMS-MODELS-AND-IMPLEMENTATION-33973-001/TxMgr-proj2.pdf>