

A Project Report On Hadoop Big Data Sort



CSC 550 1 | Big Data | Fall 2017

Under the esteemed guidance of

Dr. Da Qi Ren

INTERNATIONAL TECHNOLOGICAL UNIVERSITY

2711 N 1st St., SAN JOSE, CA 95134

ACKNOWLEDGEMENT

We would like to express our heartfelt appreciation and sincere gratitude to our Professor, Dr. Da Qi Ren for giving us this opportunity and encouraging our research.

We would also like to thank International Technology University for providing us with resources necessary for the research.

Lastly, We would like to thank our family and friends for their continued support throughout the course of my project.

Team Members

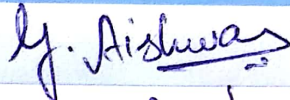
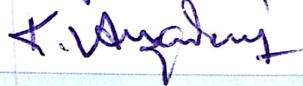
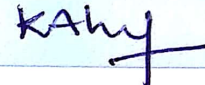

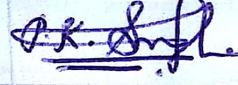
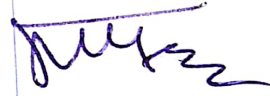
Student Name	Student ID	Signature
Aishwarya Ganesh Murthy	92436	
Amsaveni Kanagaraj	90867	
Lakshmi Kongunattumadam Ananthanarayanan	88784	
Sheela Krishnan	92737	
Sumitha P K	92065	
Sweta LNU	92752	

Table of Contents

1 Abstract	1
2 Introduction.....	2
2.1 Big Data	2
2.2 Hadoop.....	3
2.3 Map Reduce	4
2.4 HDFS	5
3. System Configuration	7
4 Data Processing Workflow	8
4.1 Our Approach	8
4.1.1 TeraGen	9
4.1.2 TeraSort	9
4.1.3 TeraValidate	10
5 Source Code.....	11
6 Execution and Output	12
7 Performance Analysis.....	14
7.1 TeraGen:.....	14
7.2 TeraSort	16
7.3 TeraValidate	18
8 Conclusion	20

9 References	21
---------------------------	-----------

List of Figures

Fig.1 Hadoop Architecture	4
Fig.2 Map Reduce Framework.....	5
Fig.3 HDFS Architecture.....	6
Fig.4 System Overview	7
Fig.5 Java Version.....	7
Fig.6 Hadoop Version	8
Fig.7 Data Processing Workflow	8
Fig.8 Data Generation logs	14
Fig.9 DataGen Folder in HDFS	15
Fig.10 Data Generation performance	15
Fig.11 Data Sorting logs	16
Fig.12 DataSort Folder in HDFS.....	17
Fig.13 Data Sorting performance	17
Fig.14 Data Validation logs	18
Fig.15 DataValidate folder in HDFS.....	19
Fig.16 Data Validation performance.....	19

1 Abstract

This project implements Hadoop MapReduce to sort 10 GB of data in HDFS using GraySort. The output of MapReduce is a sequence of files comprising of the final sorted dataset. GraySort consists of three parts: teragen, terasort and teravalidate. Teragen is used to generate the data. Terasort is used to submit a MapReduce job that sorts the data. Teravalidate is used to validate if the data has been sorted.

The Hadoop framework allows distributed processing of large data sets across clusters of computers using simple programming models. MapReduce is a programming model and an associated implementation for processing and generating large data sets. A map function can process a key/value pair to generate a set of intermediate key/value pairs, and a reduce function merges all intermediate values associated with the same intermediate key.

3Vs (volume, variety and velocity) are three defining properties or dimensions of big data. Volume refers to the amount of data, variety refers to the number of types of data and velocity refers to the speed of data processing. The three Vs of big data describe the characteristics that make big data different from other data processing. Hadoop services is a big data technology that provide data storage, data processing, data access, data governance, security, and operations.

2 Introduction

2.1 Big Data

“Big Data is a blanket term that describes large volume of data – both structured and unstructured, and technologies needed to gather, organize, process, and gather insights from large datasets. While the problem of working with data that exceeds the computing power or storage of a single computer is not new, the pervasiveness, scale, and value of this type of computing has greatly expanded in recent years.

The basic requirements for working with big data are the same as the requirements for working with datasets of any size. However, the massive scale, the speed of ingesting and processing, and the characteristics of the data that must be dealt with at each stage of the process presents significant new challenges when designing solutions. The goal of most big data systems is to surface insights and connections from large volumes of heterogeneous data that would not be possible using conventional methods.

Therefore, **Big Data** is:

- Large datasets
- The category of computing strategies and technologies that are used to handle large datasets”[1]

The three Vs of big data describe some of the characteristics that make big data different from other data processing. 3Vs (volume, variety and velocity) are three defining properties or dimensions of big data. Volume refers to the amount

of data, variety refers to the number of types of data and velocity refers to the speed of data processing.

2.2 Hadoop

Apache Hadoop is an open source software platform for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. Hadoop services provide for data storage, data processing, data access, data governance, security, and operations.[2]

The base Apache Hadoop framework is composed of the following modules:

- *Hadoop Common* – contains libraries and utilities needed by other Hadoop modules.
- *Hadoop Distributed File System (HDFS)* – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.
- *Hadoop YARN* – a platform responsible for managing computing resources in clusters and using them for scheduling users' applications.
- *Hadoop MapReduce* – an implementation of the MapReduce programming model for large-scale data processing.

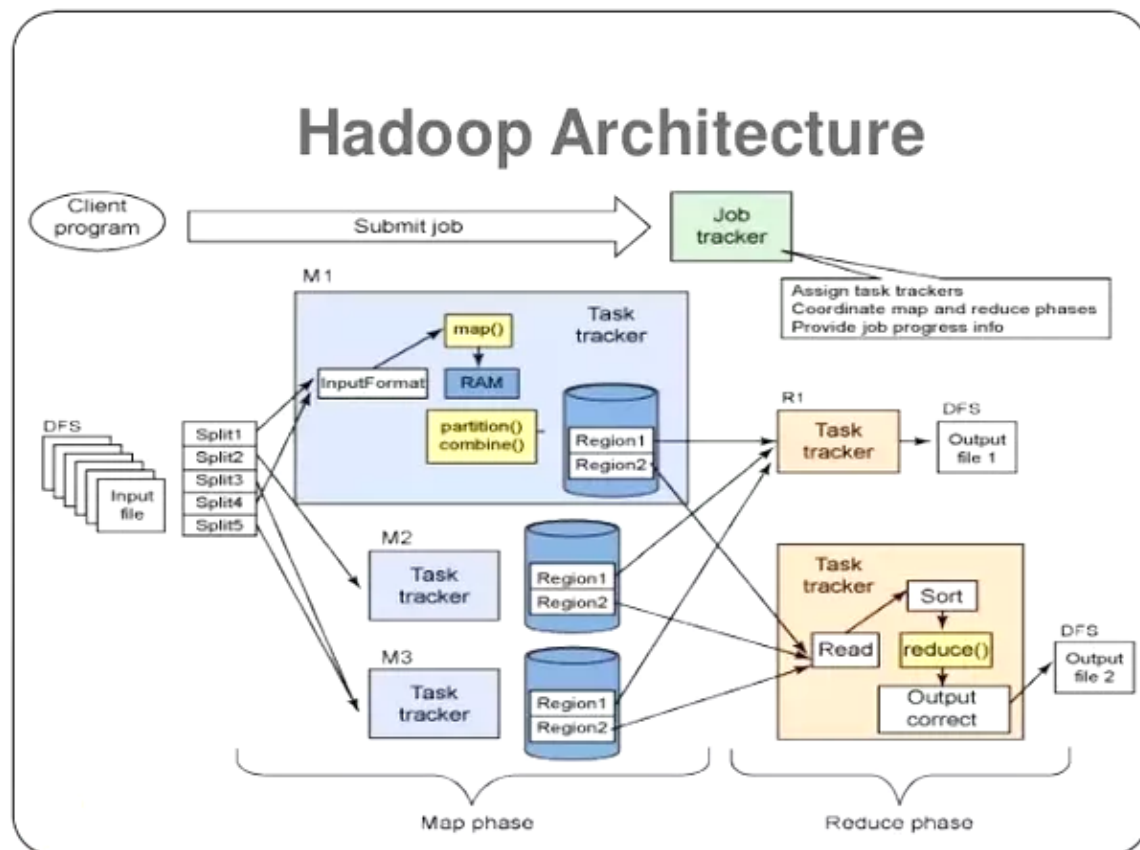


Fig.1 Hadoop Architecture

2.3 Map Reduce

The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.[3]

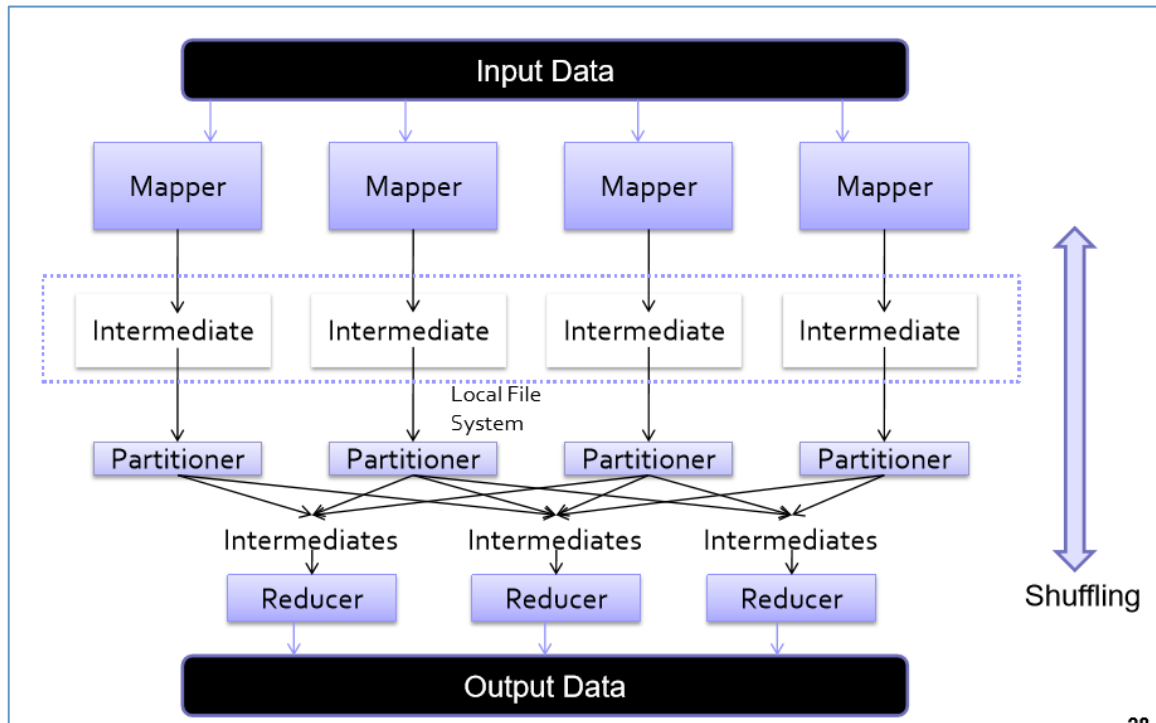


Fig.2 Map Reduce Framework

2.4 HDFS

The Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications. HDFS is a distributed file system that provides high-performance access to data across Hadoop clusters. Like other Hadoop-related technologies, HDFS has become a key tool for managing pools of big data and supporting big data analytics applications.[4]

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

Features of HDFS

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.[5]

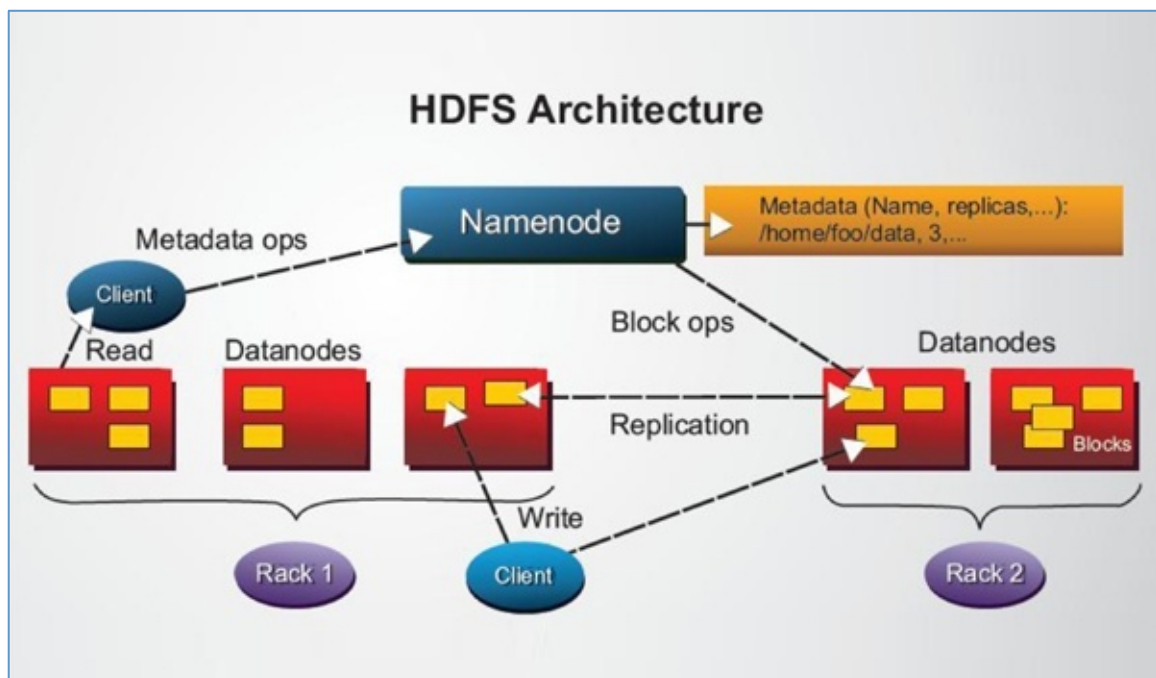


Fig.3 HDFS Architecture

3. System Configuration

- Memory: 8 GB
- Storage: 80 GB
- CPU Core: 2.3 GHz Intel Core i7
- OS: Mac OS



Fig.4 System Overview

- Java Version: 1.8.0

```
java version "1.8.0_131"  
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)  
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
```

Fig.5 Java Version

- Hadoop Version: 2.8.1

```

/usr/local/Cellar/hadoop/2.8.1/libexec/etc/hadoop/hadoop-env.sh: line 25: /Library/Java/JavaVirtualMachines/jdk1.8.0_1
31.jdk/Contents/Home: is a directory
Hadoop 2.8.1
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r 20fe5304904fc2f5a18053c389e43cd26f7a70fe
Compiled by vinodkv on 2017-06-02T06:14Z
Compiled with protoc 2.5.0
From source with checksum 60125541c2b3e266cbf3becc5bda666
This command was run using /usr/local/Cellar/hadoop/2.8.1/libexec/share/hadoop/common/hadoop-common-2.8.1.jar

```

Fig.6 Hadoop Version

4 Data Processing Workflow

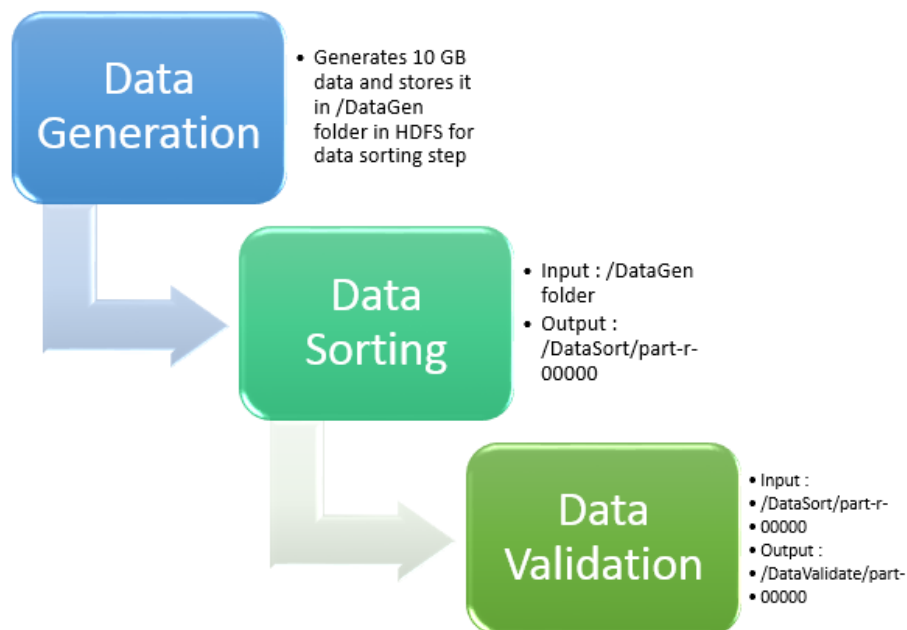


Fig.7 Data Processing Workflow

4.1 Our Approach

We have implemented Hadoop MapReduce to sort 10 GB of data in HDFS using GraySort. The output of MapReduce is a sequence of files comprising the final sorted

dataset. GraySort consists of three parts: teragen, terasort and teravalidate. Teragen is used to generate the data. Terasort is used to submit a MapReduce job that sorts the data. Teravalidate is used to validate if the data has been sorted.

4.1.1 TeraGen

TeraGen is a map/reduce program to generate 10 GB data. There are 75 map tasks and each task will generate a 128 MB. Within each file, data is pseudorandomly generated and hashed to create the keys which is used to sort in TeraSort.[6]

We can browse the HDFS by using Hadoop web interface. TeraGen data consists of 10 bytes of Key followed by 90 bytes of data. A single file contains 3 blocks and each block is 128 MB. Thus, the blocks are triple replicated inside HDFS for high availability.

Input: The number of rows and the output directory

Output Format:

The format of the output data is:

- <10 bytes key> <10 bytes rowid> <78 bytes filler> \r \n
- The keys are random characters from the set ' ' .. '~'.
- The rowid is the right justified row id as a int.
- The filler consists of 7 runs of 10 characters from 'A' to 'Z'.

4.1.2 TeraSort

TeraSort is used to sort the data. TeraSort has two phases: the map phase and the reduce phase.[7] The map phase partitions the data into 75 ranges and the reduce

task sort through the data and write it to the HDFS. Each map task samples the data and creates a set of range partitions that is used to map the records into. It doesn't sort the data yet, it just partitions the data. After the map tasks have completed, the reduce tasks start. Each reduce task is put in charge of one or more buckets of output of the map tasks. The map tasks results are stored in temporary files, they are not saved in HDFS and are not persisted beyond the life of the job. Once the task finishes, the next task in the queue begins. The number of reducers is controlled by `mapred.reduce.tasks` specified in the way you have it: `-D mapred.reduce.tasks=8` would specify 8 reducers. It reaches out to the node and asks for any of the data that fits into the bucket. Once all the data from the cluster is read, the reduce task brings that all in and sorts it and then writes it as part of the final output. All the other reduce tasks perform the same in parallel.

4.1.3 TeraValidate

TeraValidate validates the sorted output to ensure that the keys are sorted within each file. It generates 1 mapper per file that checks to make sure the keys are sorted within each file. The mapper also generates "\$file:begin", first key and "\$file:end", last key. The reducer verifies that all the start/end items are in order.

If anything is wrong with the sorted output, the output of this reducer reports the problem.

5 Source Code

The Source code, README.md and other related files are available in the following link:

<https://github.com/Sumitha123/Big-Data-Sort-using-Hadoop>

jar file is available in the following link:

<https://github.com/Sumitha123/Big-Data-Sort-using-Hadoop/tree/master/target>

6 Execution and Output

1. Install Hadoop (OSX command)

```
$ brew install Hadoop
```

2. Configure `hadoop-env.sh`, `Core-site.xml`, `mapred-site.xml` & `hdfs-site.xml`.

Before running Hadoop format HDFS. Go to `/usr/local/Cellar/hadoop/2.8.1/sbin`.

```
$ hdfs namenode -format
```

3. Hadoop requires SSH access to manage its nodes, i.e. remote machines and our local machine.

```
$ ssh localhost
```

4. Start Hadoop using the command:

```
$ /usr/local/Cellar/hadoop/2.8.1/sbin/start-dfs.sh;
```

```
/usr/local/Cellar/hadoop/2.8.1/sbin/start-yarn.sh
```

5. Using `jps` command check if all hadoop daemons are running or not.

```
$ jps
```

```
16499 Jps
```

```
73829 NodeManager
```

```
24344 SecondaryNameNode
```

```
24045 DataNode
```

```
23822 NameNode
```

```
73647 ResourceManager
```

6. Create jar file from the project directory.

Jar file can also be created from terminal by using the command:

```
$ jar -cvf Hadoop-mapreduce-graysort.jar *.*
```

7. Browse HDFS directory

<http://localhost:50070/explorer.html#/>

8. Generate 10 GB data using TeraGen (100 Million 100-byte data)

```
$ hadoop jar hadoop-mapreduce-graysort.jar teragen  
100000000 /DataGen
```

The output file generated using TeraGen is stored in /DataGen folder in HDFS.

9. Sort data using TeraSort

```
$ hadoop jar hadoop-mapreduce-graysort.jar terasort  
/DataGen /DataSort
```

This sorted output file is stored in /DataSort/part-r-00000 in HDFS.

10. Validate the output using TeraValidate

```
$ hadoop jar hadoop-mapreduce-graysort.jar teravalidate -D  
mapped.reduce.tasks=8 /DataSort /DataValidate
```

11. The output of TeraGen, TeraSort and TeraValidate can be viewed by browsing HDFS directory at:

<http://localhost:50070/explorer.html#/>

12. To clean up previous input, output folders recursively in HDFS, use the following commands:

```
$ hadoop fs -rm -r /DataGen  
$ hadoop fs -rm -r /DataSort  
$ hadoop fs -rm -r /DataValidate
```

7 Performance Analysis

7.1 TeraGen:

Time taken to generate data: 2.03 minutes

```
Map-Reduce Framework
  Map input records=100000000
  Map output records=100000000
  Map output bytes=10200000000
  Map output materialized bytes=10400000450
  Input split bytes=8025
  Combine input records=0
  Combine output records=0
  Reduce input groups=100000000
  Reduce shuffle bytes=10400000450
  Reduce input records=100000000
  Reduce output records=100000000
  Spilled Records=395957882
  Shuffled Maps =75
  Failed Shuffles=0
  Merged Map outputs=75
  GC time elapsed (ms)=5321
  Total committed heap usage (bytes)=37643354112
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=10000000000
File Output Format Counters
  Bytes Written=10000000000
17/11/11 14:07:00 INFO terasort.TeraSort: done
```

Fig.8 Data Generation logs

localhost:50070/explorer.html#/DataSort

Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities

Browse Directory

Show 25 entries
Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	suneelsharma	supergroup	0 B	Nov 11 14:06	3	128 MB	._SUCCESS
-rw-r--r--	suneelsharma	supergroup	0 B	Nov 11 13:55	10	128 MB	._partition.list
-rw-r--r--	suneelsharma	supergroup	9.31 GB	Nov 11 14:06	3	128 MB	part-r-00000

Showing 1 to 3 of 3 entries

Hadoop, 2017.

Fig.9 DataGen Folder in HDFS

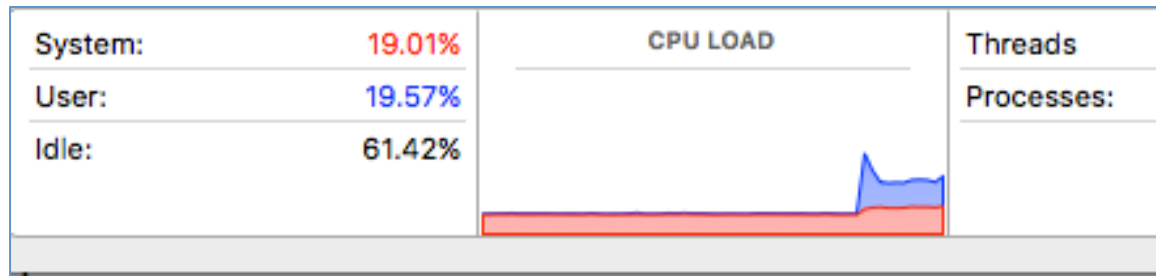


Fig.10 Data Generation performance

7.2 TeraSort

Time taken to sort data: 11.28 minutes

```
Map-Reduce Framework
  Map input records=100000000
  Map output records=100000000
  Map output bytes=10200000000
  Map output materialized bytes=10400000450
  Input split bytes=8025
  Combine input records=0
  Combine output records=0
  Reduce input groups=100000000
  Reduce shuffle bytes=10400000450
  Reduce input records=100000000
  Reduce output records=100000000
  Spilled Records=395957882
  Shuffled Maps =75
  Failed Shuffles=0
  Merged Map outputs=75
  GC time elapsed (ms)=5321
  Total committed heap usage (bytes)=37643354112
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=10000000000
File Output Format Counters
  Bytes Written=10000000000
17/11/11 14:07:00 INFO terasort.TeraSort: done
```

Fig.11 Data Sorting logs

localhost:50070/explorer.html#/DataSort

Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities

Browse Directory

/DataSort
Go

Show 25 entries
Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	suneelsharma	supergroup	0 B	Nov 11 14:06	3	128 MB	._SUCCESS
-rw-r--r--	suneelsharma	supergroup	0 B	Nov 11 13:55	10	128 MB	._partition.list
-rw-r--r--	suneelsharma	supergroup	9.31 GB	Nov 11 14:06	3	128 MB	part-r-00000

Showing 1 to 3 of 3 entries
Previous
1
Next

Hadoop, 2017.

Fig.12 DataSort Folder in HDFS

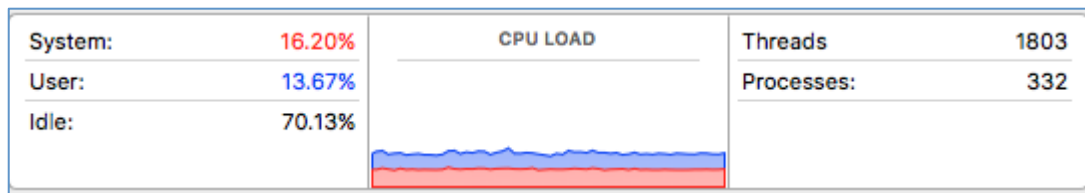


Fig.13 Data Sorting performance

7.3 TeraValidate

Time taken to validate data: 2.17 minutes

```
File System Counters
  FILE: Number of bytes read=604436
  FILE: Number of bytes written=1264801
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=20000000000
  HDFS: Number of bytes written=25
  HDFS: Number of read operations=13
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=4
Map-Reduce Framework
  Map input records=100000000
  Map output records=3
  Map output bytes=83
  Map output materialized bytes=95
  Input split bytes=108
  Combine input records=0
  Combine output records=0
  Reduce input groups=3
  Reduce shuffle bytes=95
  Reduce input records=3
  Reduce output records=1
  Spilled Records=6
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=219
  Total committed heap usage (bytes)=440401920
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=10000000000
File Output Format Counters
  Bytes Written=25
```

Fig.14 Data Validation logs

Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities

Browse Directory

Show 25 entries
Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	suneelsharma	supergroup	0 B	Nov 11 14:17	3	128 MB	_SUCCESS <input type="button" value="🗑"/>
-rw-r--r--	suneelsharma	supergroup	25 B	Nov 11 14:17	3	128 MB	part-r-00000 <input type="button" value="🗑"/>

Showing 1 to 2 of 2 entries

Hadoop, 2017.

Fig.15 DataValidate folder in HDFS

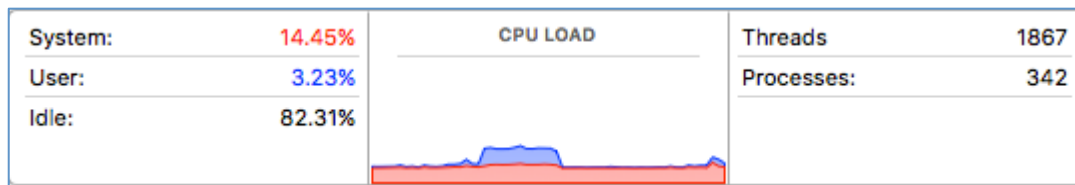


Fig.16 Data Validation performance

8 Conclusion

Hadoop performance is sensitive to the Hadoop version used, HDFS, JVM, Operating System, Network bandwidth and the underlying hardware.

We have implemented high performance gray sort that generates, sorts and validates data with relatively low latency. The time taken to sort 10 GB data was approximately 11.28 minutes, which is less compared to other traditional big data sorting models. Every Hadoop version is distributed with a very large set of configuration parameters, and a rather large subset of these parameters can potentially impact performance. The Java heap size requirements, the number of MapReduce task, the number of hardware cores, the input/output bandwidth, as well as the amount of available RAM impact the basic performance of a big data processing system.

9 References

- [1]<https://www.digitalocean.com/community/tutorials/an-introduction-to-big-data-concepts-and-terminology>
- [2]<https://hortonworks.com/apache/hadoop/>
- [3]https://www.tutorialspoint.com/hadoop/hadoop_mapreduce.htm
- [4]<http://searchbusinessanalytics.techtarget.com/definition/Hadoop-Distributed-File-System-HDFS>
- [5]https://www.tutorialspoint.com/hadoop/hadoop_hdfs_overview.htm
- [6]<https://hadoop.apache.org/docs/r1.0.4/api/org/apache/hadoop/examples/terasort/>
- [7] Owen O'Malley, 2008. "TeraByte Sort on Apache Hadoop".
- [8] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December 2004.