International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

# Performance Evaluation of Hadoop Distributed File System and Local File System

**Linthala Srinithya[1], Dr. G. Venkata Rami Reddy[2]**

[1]M.Tech. Student, School of Information Technology, JNTUH, Hyderabad, India

[2]Associate Professor, School of Information Technology, JNTUH, Hyderabad, India

**Abstract:** ***Hadoop is a framework which enables applications to work and petabytes of data on large clusters with thousand of nodes built of commodity hardware. It provides a Hadoop Distributed File System (HDFS) that stores data on the computed nodes, providing very high aggregate bandwidth across the cluster. In addition, Hadoop implements a parallel computational paradigm named map-reduce which divides the application into many small segments of work, each of which may be executed or re executed on any node in the cluster. In this project I would like to analyze performance of HDFS and LFS with respective read and write. To measure the performance I will set up a hadoop cluster and design an interface which gives us the size of the file, time taken for upload or download from Local File System(LFS) and Hadoop Distributed File System(HDFS). By literature survey the expected HDFS writing performance scales will on both small and big data set were it is however lower than on the small data-set. This work also draws a comparison between the HDFS (Hadoop Distributed File System) and LFS performances.***

**Keywords:** HDFS, LFS, DFS, Read, Write, Update, Commodity hardware, fault-tolerant.

## 1. Introduction

Storage and accessing of data are most important things in computer world. The data can be stored permanently in secondary storage device in the form of a "**File**". There are several operation (creating reading, writing, etc.) required to store and maintain the data in files. All these operations are provided by **"File System".**

The file system is a type of data store which can be used to store, retrieve and update a set of files. The file system has maintained aspects such as file names, directories, space management, and meta-data. Each operating system is having its own file system to maintain files in file or directories structure. There are several file systems available.
A. Disk File System/Local File Systems
B. Distributed File Systems(DFS)

### A. Disk File System/Local File Systems

Disk File System/Local File Systems is generally provided by operating systems. It deals with organizing the files on local disk. The data stored in local disk can be accessed with-in the machine only. The data has to be physically copied into another disk or the total file has to be transferred to other machine for accessing.

Centralized server is the solution to share the files with different computers. In this technique, the data is stored in a dedicated file server and other nodes can access that data through network. To create this type of environment, high configuration is required to maintain centralizes server to store large data. If the server is failed, any host cannot access data to complete their task. Of course, the back-up can be maintained for server. Another problem with centralized server is heavy load on server. The server has to take the request from all client machines and increasing the clients will affect the performance.

### B. Distributed File Systems (DFS)

The solution to the problems with centralized server is distributed file system. In this, the data will be distributed to number of servers so that the data-process load will be distributed to several servers instead of one server. The replication can be maintained in distributed systems so that the failure of one server will not affect the file system. Since the data can be retrieved from different servers, it can be downloaded in parallel fashion.

### 1.1 Motivation

There are several file systems available to store and process large data. The user has to select one of the suitable files systems among the available file system. The user may face some trouble in this selection to choose a suitable file system.
There are various aspects to compare:

- Writing Time
- Reading Time
- Concurrent access of file etc.

Recently much work is going on HDFS (Hadoop Distributed File System). Since it is fault tolerant distributed file system and also supports parallel programming by using map-reduce and frameworks. The HDFS handles well for large data. Hence the motivation behind the project is to verify whether the read and write performance of HDFS is efficient for small and large files than LFS or not. If HDFS supports well for large files I can further use HDFS for my future works related to distributed file system.

### 1.2 Problem Statement

Verify and compare the performance of HDFS and LFS access time for various file sizes ranging from small file sizes to large file sizes

### 1.3 Objectives of Project

- Build a HDFS cluster environment with one machine dedicated to name-node, one machine for secondary-node and 3 machines for data-nodes.
- The main objective of my project is to verify and compare the reading and writing performance of files on "LFS" and "HDFS".
- Design and implement the separate read models for reading files from both HDFS and LFS.
- Integrate the both read models into 1 GUI.
- Design and implement and separate write model for writing files from both HDFS and LFS.
- Integrate the both write models into 1 GUI
- Generating the graph by using access times and file sizes of both read write interfaces for performance analysis.

### 1.4 Limitation of the Project

The following are few limitations of my project.

- HDFS cluster is established in LAN and not yet tested on WAN.
- The Cluster is tested only on limited missions.
- The tested files are from sizes 1MB-1GB.
- The performances of map-reduce applications are not considered.

## 2. Analyses

### 2.1 Software Requirement Specification

#### 2.1.1 Purpose

The purpose of this project is to evaluate and compare the read and write performance of LFS (Local File System) and HDFS (Hadoop Distributed File System).

#### 2.1.2 Scope

In this project, the read and write operations are considered from the end user. This project can be used in both pseudo mode and in fully distributed environments. The interface can be installed on any data node or in client machine. The performance of map-reduce application, and other strategies like catching and replication are not evaluated.

### 2.2 Non-Functional Requirement

#### 2.2.1 Performance

Since this project is related to HDFS (Hadoop Distributed File System) performance evaluation with respective to input/output, this project needs to give high through put for big files. The HDFS architecture itself supports big data by storing the big files into number of blocks (64MB size each) and supports map-reduce frame work. So that our system can give high performance.

#### 2.2.2 Availability

Whenever user wants to access HDFS (Hadoop Distributed File System), he is able to get required data enhanced, availability is provided in our systems.

#### 2.2.3 Scalability

The system should be able to provide the new resources. Our system is to be designing by considering this future. Hence, our system will be scalable.

#### 2.2.4 Reliability

Our system performs its functions well in normal cases as well as un-expected circumstances because HDFS (Hadoop Distributed File System) provides fault-tolerant future. Hence our system is able to maintain reliability even through failure happens.

#### 2.2.5 User Requirements

- HDFS (Hadoop Distributed File System) Cluster (pseudo/distributed) should be built.
- User interface should be implemented for uploading and downloading the files easily from form(to be implemented)
- Users are able to give various files with sizes from 1MB-1GB for upload/download to/from the LFS/HDFS.

#### 2.2.6 Software Requirements

- Linux/Ubuntu Operating System
- Java Software Development Kit with JXL package
- HDFS
- Office Suite (Spread sheet for graph)

#### 2.2.7 Hardware Requirements

Five computes with minimum configuration of Pentium IV, 4GB RAM, 500GB HDD(Hard Disk Drive).

- System for Name-Node
- System for Secondary-Node
- Systems for Data-Node.

Interconnected network for this five computer with LAN /Internet.

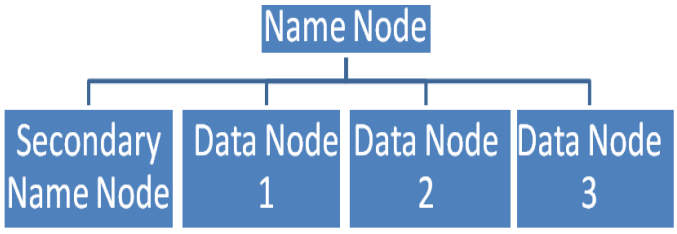


**Figure 1:** Diagram of HDFS Cluster

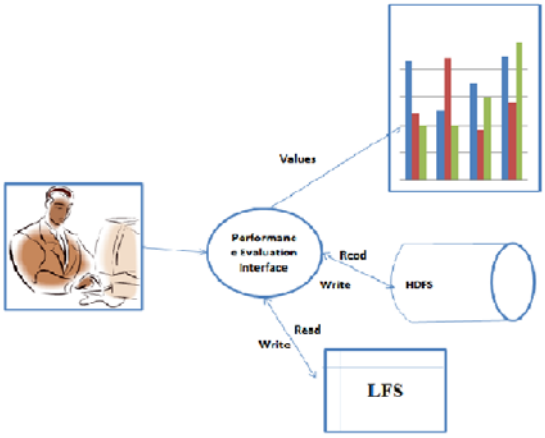


**Figure 2:** Performance Evaluation System

### 2.3 Algorithms

There are four algorithms required to design interface.

1. Writing to LFS
2. Reading from LFS
3. Writing to HDFS
4. Reading from HDFS

#### 2.3.1 Writing to LFS Algorithm

This algorithm writes the data to LFS and calculates the time taken to write. Since it is writing large data, which should be available in secondary storage. The algorithm takes the data from one file called source file and write to another file called destination file.

This algorithm initially prompts the user to give the source file and destination file. It opens the files for given source file name and destination file name for reading and writing respectively. After opening required files successfully, it creates a time object "T1" that initialized to current system time. It starts reading the data byte by byte using the given streams from source file and write the same to destination file until it finds end of file in source file. After reading and writing data from LFS, this algorithm creates another time object "T2" with current system time. The difference between T1 and T2 is the time taken to write the data on LFS and it is stored in "T". Finally, send the file size, time taken to the spread sheet.

**The algorithm is given below.**

Step1: Import the source file name and destination file name.
Step2: Open the source file and destination file name LFS.
Step3: Store the system timings T1.
Step4: Write the bytes from given source file to destination in
LFS
Step5: Repeat step4 until end of file of Source file.
Step6: Store the system time in T2.
Step7: Find the difference between T1 and T2 let it be T.
Step8: Store the size of file, value of d to Excel sheet in proper cells.

#### 2.3.2 Reading from LFS Algorithm

This algorithm read the data to LFS and calculates the time taken to write. The algorithm takes the data from the file called source file.

This algorithm initially prompts the user to give the source file. It opens the files for given source file for reading. After opening required files successfully, it creates a time object "T1" that initialized to current system time. It starts reading the data byte by byte using the given streams from source file until it finds end of file in source file. After reading data from LFS, this algorithm creates another time object "T2" with current system time. The difference between T1 and T2 is the time taken to read the data on LFS and it is stored in "T". Finally, send the file size, time taken to the spread sheet.

The algorithm is designed as following:

Step1: Import the source file name
Step2: Open the source file in LFS.
Step3: Store the system timings T1.
Step4: Read the bytes from given source file from LFS
Step5: Repeat step4 until it finds end of file.
Step6: Store the system time in T2.
Step7: Find the difference between T1 and T2 let it be T.
Step8: Store the size of file, value of d to Excel sheet in proper cells.

#### 2.3.3 Writing to HDFS Algorithm

This algorithm writes the data to HDFS and calculates the time taken to write. Since it is writing large data, which should be available in secondary storage in LFS. The algorithm takes the data from one file called source file in LFS and write to another file called destination file HDFS.

This algorithm initially prompts the user to give the source file and destination file. It opens the files for given source file name in LFS and destination file name in HDFS for reading and writing respectively. After opening required files successfully, it creates a time object "T1" that initialized to current system time. It starts reading the data byte by byte using the given streams from source file and write the same to destination file until it finds end of file in source file. After reading and writing data from LFS and HDFS, this algorithm creates another time object "T2" with current system time. The difference between T1 and T2 is the time taken to write the data on HDFS and it is stored in "T". Finally, send the file size, time taken to the spread sheet.

**The algorithm is designed as following:**

Step1: Import the source file name and destination file name
Step2: Open the source file and destination file name in LFS.
Step3: Store the system time in Time object "T1"
Step4: Write the Given File Source File from LFS to destination File in HDFS.
Step5: Repeat step4 until end of file in Source file.
Step6: Store the system time in T2.
Step7: Find the difference between T1 and T2 let it be T.
Step8: Store the size of file, value of d to Excel sheet in proper cells.

#### 2.3.4 Reading from HDFS Algorithm

This algorithm read the data to HDFS and calculates the time taken to write. The algorithm takes the data from the file called source file.

This algorithm initially prompts the user to give the source file. It opens the files for given source file for reading. After opening required files successfully, it creates a time object "T1" that initialized to current system time. It starts reading the data byte by byte using the given streams from source file until it finds end of file in source file. After reading data from LFS, this algorithm creates another time object "T2" with current system time. The difference between T1 and T2 is the time taken to read the data on HDFS and it is stored in

"T". Finally, send the file size, time taken to the spread sheet.

**The algorithm is designed as following:**
Step1: Import the source file name
Step2: Open the source file in HDFS.
Step3: Store the system timings T1.
Step4: Read the bytes from given source file from HDFS
Step5: Repeat step4 until it finds end of file.
Step6: Store the system time in T2.
Step7: Find the difference between T1 and T2 let it be T.
Step8: Store the size of file, value of d to Excel sheet in proper cells.

### 2.4 Conclusion on Analysis

In this I have reviewed the basic architecture of HDFS and Read and Write operations in detail. There are various functional requirements like interface, read and write functions and graph generation are described. The non-functional requirements such as performance, scalability, reliability, availability, user requirements, software requirements and hardware requirements are also explained related to my project. Algorithms and flow charts are also designed for read and write operations for both read and write operations.

## 3. Design

I have designed read and write operations both Local File System and Hadoop Distributed File System after building HDFS cluster. The various modules for both read and write operations are designed with the help of required diagrams. In addition to these modules I have also designed a user interface for performance evaluation of read and write operations for both Local File System and HDFS.

### 3.1 Modules

There are 5 Modules in my project.

1. Building HDFS cluster
2. Write Module
3. Read Module
4. Interface
5. Generating Graphs for evaluation.

#### 3.1.1 Building HDFS Cluster

There are several phases to build HDFS Cluster.

1. Pre-requites
2. Configuration HDFS
3. Start/Shutdown cluster.

One of the first tasks in hadoop deployment is selecting the distribution and version of hadoop that is most appropriate given the features and stability required. It is available directly from Apache in both source and binary formats. It provides not just the distributed file system, but also the map reduce processing framework—many users view it as the core of a larger system. In this sense, hadoop is analogous to an operating system kernel, giving us the core functionality upon which we build higher-level systems and tools. Many of these related libraries, tools, languages and systems are also open source projects available from the ASF.

There are several versions available for HDFS. I have chosen hadoop 0.22.0release for my project. The hadoop community released version 0.22, which was based on trunk was released after 0.23 with less functionality. This was due to when the 0.22 branch was cut from trunk. Hadoop 0.22.0 features:

- HBase support with hflush and hsync.
- Symbolic links
- Backup node and checkpoint node
- Hierarchical job queues.
- Job limits for Queue/pool
- Dynamically stop/start job queue
- Advances in new map-reduce API: Input/output formats, chain mapper/reducer.
- Task tracker black listing
- Distributed Cache sharing.

Picking the right hardware is critical. One of the major advantages of hadoop is its ability to run on so-called commodity hardware. Hadoop hardware comes in two distinct classes: masters and workers. Master nodes are typically more robust to hardware failure and one critical cluster services. Loss of a master almost certainly means some kind of service disruption. On the other hand, worker nodes are expected to fail regularly. This directing impact the type of hardware as well as the amount of money spent on these two classes of hardware. Clusters with fewer than 20 workers nodes-do not require much for master nodes in terms of hardware. A solid base line hardware profile for a cluster of this size is a dual quad core 2.6 Ghz CPU, 24Gb of DDR3 Ram, dual 1 GB Ethernet NIC's, a SASdrive controller, and atleast two SATA-II drives in a capital JBOD configuration in addition to the host OS device.

The Secondary name node is almost always identical to the name node. Not only does it require the same amount of RAM and Disk, but when absolutely everything goes wrong, it wins up being the replacement hardware for the name node. When sizing worker machines for hadoop, there are a few points to consider given the each worker node in a cluster is responsible for both storage and computation, we need to en-source not only that there is enough storage capacity, but also that we have the CPU and memory to process that data.

Once, the hardware for masters and worker nodes is selected, the size of the cluster has to be fixed in my project I have to build the HDFS cluster and 5 nodes. There are two types of installation for HDFS.

- Pseudo Distribution
- Fully Distribution.

In Pseudo Installation, the name node and data node are installed on one machine. In Fully Distributed Installation, there will be a dedicated machines for name node, secondary name node and separate machines for data nodes.

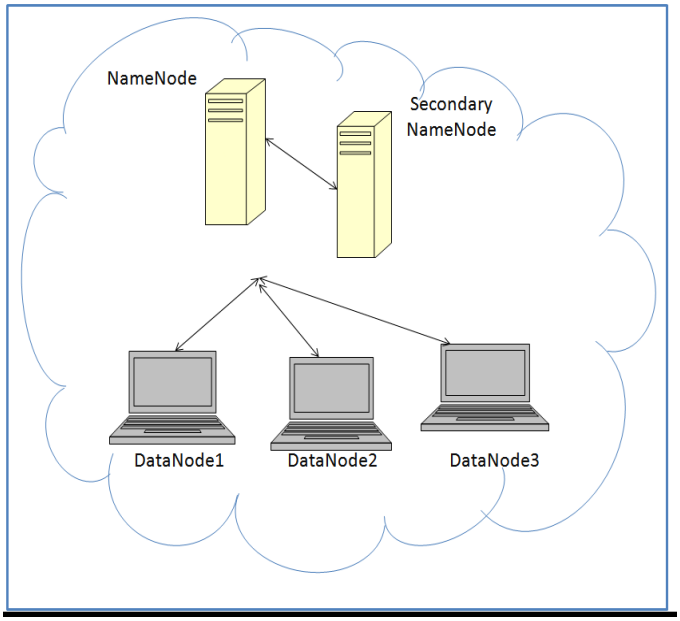


**Figure 3:** HDFS Cluster

(1 Name Node, 2 Secondary Name Nodes, 3 Data Nodes)

After fixing the cluster size, OS has to be installed on both master and marker nodes and assign the IP address to nodes as shown in figure. Hadoop is designed on Java Development Kit has to be installed on each system.
Once the hardware and software for nodes are fixed and installed, the selected hadoop release can be installed on master and worker machines. After installing hadoop on all nodes, some of the system files have to be configured to build the cluster and start the hadoop cluster.

After creation of cluster, I have to implement an interface to upload or download file between HDFS and local disk including the time calculations and also stored the values separately for designing graph.

### 3.1.2 Write Module

This module is designed to create the environment to write the data on to LFS and HDFS. The separate functions have to be developed for LFS and HDFS.m The write module prompts for source file and destination file. There are 2 cases in write module which includes writing to local disk, writing to HDFS. The write function has to copy/ upload to the corresponding file system and calculate the time taken for that process. Finally the size of the file and calculated time should be sent to spread sheet at proper locations.

### 3.1.3 Read Module

This module is designed to create the environment to read the data from LFS and HDFS. The separate functions have to be developed for LFS and HDFS. The read module prompts for source file. There are 2 cases in read module which includes reading to local disk, writing to HDFS. The read function read the data from corresponding file system and calculates the time taken for that process. Finally the size of the file and calculated time should be sent to spread sheet at proper locations.

### 3.1.4 Designing Interface

I have to design an interface to read and write data. The interface should provide the environments to read and write files to/from LFS and HDFS. There are two interfaces to be designed; one is for writing data into local file system and HDFS another for reading data from LFS as well as HDFS. This provides an environment to take file names as input and call the read and write modules by passing the given file names as arguments.

## 4. Implementation

### 4.1 Building HDFS Cluster

Building of HDFS cluster is discussed. The name-node is mentioned as “Master” and data-nodes are called “Slaves”.
HDFS Cluster Setup:
Phase-I : Pre-requisites
Phase-II : Configuration HDFS
Phase-III: Starting/Shutdown of HDFS cluster.

### 4.2 Key Functions

There are various functions to be implemented to achieve the objectives.

- Write Function
- Read Function
- Generating Graph
- Interface design

**Function to write data to HDFS:**
This function read the data from source file specified in source file TextField of LFS and write to destination file specified in destination file Test Field of HDFS. It calculates the time to write data to HDFS and call a “copyToExcel ()” function to store the value in spread sheet to generate graph.

**Function to Write data to LFS:**
This function read the data from source file specified in source file TextField of LFS and write to destination file specified in destination file Test Field of LFS. It calculates the time to write data to HDFS and call a “copyToExcel ()” function to store the value in spread sheet to generate graph.

**Function to Read data from HDFS:**
This function read the data from source file specified in source file TextField of HDFS. It calculates the time to write data to HDFS and call a “copyToExcel ()” function to store the value in spread sheet to generate graph.

**Function to Read data from LFS:**
This function read the data from source file specified in source file TextField of LFS. It calculates the time to write data to HDFS and call a “copyToExcel ()” function to store the value in spread sheet to generate graph.

### 4.3 Generating Graph

I have implemented a function to store the values to spread sheet to generate the graph. This function accepts various arguments to place the value ate a particular cell. The

Boolean argument "HADOOP" specifies whether the test is for HDFS or LFS. The argument file specifies the name of the spread sheet with path. Size argument specifies the file size. Test argument gives the test numbers.

### 4.4 Designing Interface

I have implemented the interface by using java swings it includes the key function implemented in previous section. It gives a user friendly environment to input the file names and to test the read and write time. The corresponding code can be explained below:

### 4.5 Interface for writing

The interface is designed with two (2) classes HadoopWrite extends JPanel and Write Test extends JFrame. HadoopWrite defines write methods for LFS and HDFS. It provides common GUI for write operation the type of the operation is given as parameter of constructor. Write Test creates objects of type HadoopWrite for both LFS and HDFS and place them in common form.

### 4.6 Interface for Reading

The interface is designed with two (2) classes HadoopRead extends JPanel and ReadTest extends JFrame. HadoopRead defines read methods for LFS and HDFS. It provides common GUI for read operation. The type of the operation is given as parameter of constructor. ReadTest creates objects of type HadoopRead for both LFS and HDFS and place them in common form.

## 5. Result Analysis

I have verified the write and read performance of both HDFS and LFS on various file sizes 1MB, 2MB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256MB, 512MB, and 1GB.

### 5.1 Write Operation

The table of values obtained for above input file sizes is shown below:

**Table 1:** Input File Sizes for Write Operation

| *File Size* | *LFS in Seconds* | *HDFS in Seconds* |
|---|---|---|
| 1 | 3.02 | 0.64 |
| 2 | 5.99 | 1.18 |
| 4 | 16.34 | 2.38 |
| 8 | 31.3 | 4.58 |
| 15 | 55.57 | 12.16 |
| 31 | 117.38 | 26.16 |
| 61 | 237.01 | 57.34 |
| 122 | 454.1 | 115.56 |
| 244 | 950.92 | 229.95 |
| 488 | 1562.52 | 387.01 |
| 977 | 3531.36 | 767.86 |

In the above table 1, I observed that for 1MB input file, LFS is taking 3.02 seconds and HDFS I respectively. It is also 4 times more for LFS than HDFS. I observed that all the values are giving similar results. In the average, LFS is taking 4.8 times compare to HDFS.

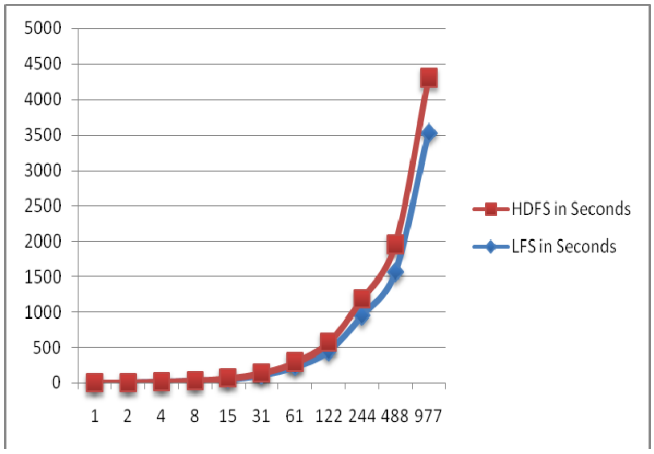


**Figure 4:** Write Performance Graph

The performance if write operation for the files with various sizes specifies above in shown Figure.

The dotted line represents the average of 3 test values taken for writing the files into LFS and solid line indicates the performance of HDFS.

By considering this graph, I have observed that the write performance for small files is not having much difference between LFS and HDFS. There is the considerable difference in performances of LFS and HDFS for large files.

### 5.2 Read Operation

The table of values obtained for above input file sizes is shown below.

**Table 2:** Input File Sizes for Read Operation

| *File Size* | *LFS in Seconds* | *HDFS in Seconds* |
|---|---|---|
| 1 | 1 | 0.16 |
| 2 | 1.94 | 0.39 |
| 4 | 3.61 | 0.64 |
| 8 | 5.97 | 1.01 |
| 15 | 12.67 | 2.6 |
| 31 | 25.59 | 3.93 |
| 61 | 63.77 | 7.9 |
| 122 | 98.06 | 14.27 |
| 244 | 194.9 | 28.75 |
| 488 | 388.84 | 50.37 |
| 977 | 767.82 | 111.8 |

In the above table 2, I observed that for 1MB input file, LFS is taking 1 second and HDFS is taking 0.16 seconds for writing and the difference is 0.84 seconds. LFS is taking more than 6 times than HDFS. For 64 MB (approximately) file, the values are 63.77 seconds and 7.9 seconds for LFS and HDFS respectively. It is also 8 times more for LFS than HDFS. I observed that all the values are giving similar results. In the average LFS is taking 6 times compare to HDFS.

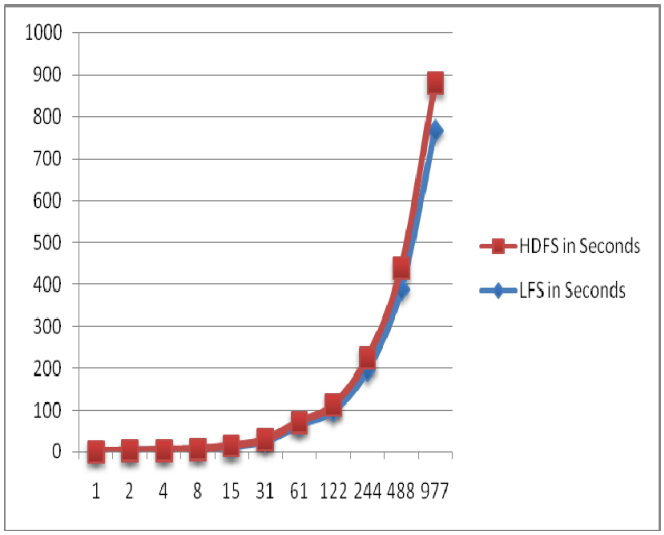


**Figure 5:** Read Perofrmance Graph

The performance of read operation for the files with various sizes specified above is shown in figure.

The blue color line represents the average of 3 test times taken for reading the data from LFS and red line indicates the performance for small files is not having much difference between LFS and KDFS. There is the considerable difference in performances of LFS and HDFS for large files.

### 5.3 Conclusion of Result Analysis

In this project, I built a HDFS cluster on 5 systems (1 Name-Node, 1 Secondary Name-Node, 3 Data-Nodes). The required key functions file read (), file write (), hdfsread (), hdfswrite () are implemented to achieve the main objective of my project. An interface is also implemented for end user to give various sizes of files for reading and writing data to/from LFS and HDFS. A graph is generated with the values taken from the interfaces and analyzed that the read and write performances of HDFS are good compare to read and write operations of LFS.

## 6. Testing

Even the project is organized well and implemented in proper manner; there may be chance of some errors or bugs. I have to verify each and every module whether it is working properly or not for improving quality if project.

### 6.1 Design of Test Cases and Scenarios

There are 5 modules in my project. In each module, we have to verify whether it is functioning properly or not. The end user may give wrong data that has to be handled by the application.

Let us verify the quality of my project by designing various test cases. I have to design various test cases for all modules. The following are the test cases.

#### 6.1.1 Test Case: 01

Test Case Name: Starting HDFS Cluster

**Description:**
This test case is designed to verify whether the HDFS cluster is started or not properly. Name-Node, Secondary-Name-Node, 3 data nodes have to be verified whether they are communicating with cluster or not.

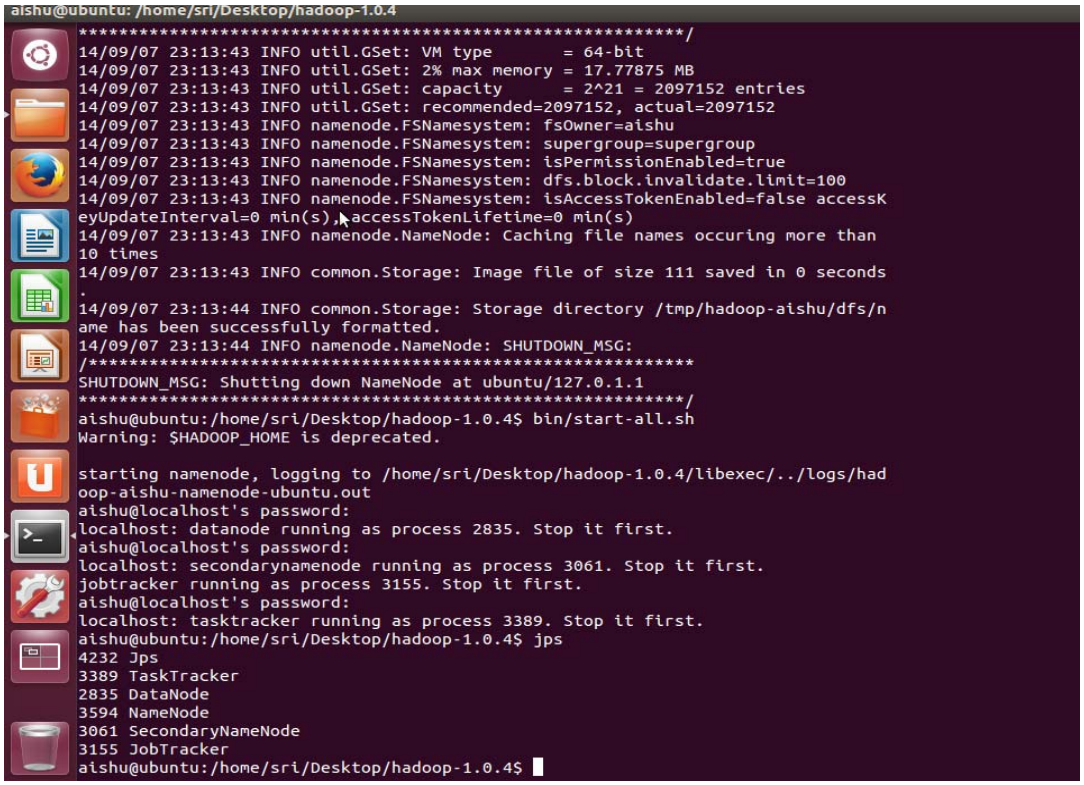


**Figure 6:** Screen Shot

**Table 3:** HDFS Cluster Test Cases

| *TID* | *Description* | *Input* | *Expected Output* | *Actual Output* | *Pass /Fail* |
|---|---|---|---|---|---|
| 1 | Verify the starting of Name Node and Data Nodes | $bin/ start-all.sh | Master: starting name node, | Master: starting name node | Pass |
| | | | Slave1: starting Secondary name node | Slavel1: starting secondary name node | Pass |
| | | | Slave2: starting data node | Slave2: starting data node | Pass |
| | | | Slave2:starting data node | Slave2:starting data node | Pass |
| | | | Slave2: starting data node | Slave2: starting data node | Pass |
| 2 | Verify JPS comma-nd in Name Node | $jps | 14399 Name-Node 12215 jps | 14399 Name-Node 12215 jps | Pass |
| 3 | Verify JPS comma-nd in Secondary Name Node | $jps | 11612 jps 16312 Secondary Name node | 11612 jps 16312 secondary name node | pass |
| 4 | Verify JPS command in Data Node | $jps | 11501 Data-Node 11612 jps | 11501 Data-Node 11612 jps | Pass |

### 6.1.2 Test Case 02

Test Case Name: Verifying Read Interface

**Description:**
The test case is designed to verify the read interface. It verifies whether the proper values or messages are displaying or not for the given source file or not. If the given file name and path are correct, the time taken to read will be displayed otherwise, it displays the error message.

**Table 4:** Read Test Cases

| *TID* | *Description* | *Input* | *Expected Output* | *Actual output* | *Pass /Fail* |
|---|---|---|---|---|---|
| 1 | Verify the result for given wrong source file name in HDFS | Source File: HDFS:/ur/ 16 MB.zip | Message box should be displayed with error message | Message box displayed | Pass |
| 2 | Verify the result for given wrong file name is LFS | Source File:/HDF Scluster/d esktop/So urce/16 MB.zip | Message box should be displayed with error message | Message bodx display | pass |
| 3 | Verify the result for reading given source file name is HDFS | Source File:/HDF S:/User/16 MB.zip | Time in Milliseconds should be displayed | 3812 | Pass |
| 4 | Verify the result for given source file name is LFS | Source File:/ HDFSclus ter/deskto p/source/1 6MB.zip | Time in Milliseconds should be displayed | 137824 | Pass |

### 6.1.3 Test Case 03

Test Case Name: Verifying the write interface.

**Description:**
This test case is designed to verify the write interface. It verifies whether the proper values or messages are displaying or not for the given source file and destination file or not. If the given file name and path are correct, the time taken to read will be displayed otherwise, it displays the error message.

**Table 5:** Write Test Case

| *TID* | *Description* | *Input* | *Expected Output* | *Actual Output* | *Pass/ Fail* |
|---|---|---|---|---|---|
| 1 | Verify the result for given wrong source file name in HDFS | Source File:/hdfscluste r/desktop/sourc e/16MB.zip Destination file: HDFS:/ur/16M B.zip | Message box should be displayed with expectation | Message Box displayed | Pass |
| 2 | Verify the result for given wrong file name is LFS | Source File:/hdfscluste r/desktop/sourc e/16MB.zip Destination file: /HDFScluster/d esktop/destinati on/16MB.zip | Message box should be displayed with expectation | Message Box displayed | Pass |
| 11 | Verify the result for given source file name is HDFS | Source File:/hdfscluste r/desktop/sourc e/16MB.zip Destination file: HDFS:/ur/16M B.zip | Time in Milliseconds should be displayed | 9737 | Pass |
| 3 | Verify the result for given source file name in LFS | Source File:/hdfscluste r/desktop/sourc e/16MB.zip Destination file: /HDFScluster/d esktop/destinati on/16MB.zip | Time in Milliseconds should be displayed | 51315 | Pass |

### 6.2 Conclusion on Testing

In this various test cases are designed for testing HDFS cluster, read interface and write interface. The HDFS is tested whether it is starting properly or not by using bin/start-all.sh and jps command. The read interface is tested by giving wrong file name in source and correct file names for both HDFS and LFS. The write interface is tested by giving wrong file names for source and destination for HDFS and LFS. My project is passed in all these tests.

## 7. Conclusion & Future Enhancement

### 7.1 Conclusion

Apache provides a distributed system hadoop Distributed File System for maintaining large data. HDFS gives an efficient environment for storing and handling large files in distributed manner compare to Local File System. LFS is able to handle small files efficiently. But for large files it takes lot of time for writing and reading. HDFS handles large files when compared to LFS. HDFS and LFS give the performance similar for writing small files and there is a notable difference in writing large files. The LFS gives better performance for small files compare to HDFS. Because the small file is available in disk and can be read well. HDFS gives less performance for reading small files because of extra load for java environment. HDFS gives better performance for reading large files because the data is divided into blocks which can be read in parallel fashion where LFS reads the data in Sequential fashion. All the read and write values obtain through interface are included and analyzed by generating graphs.

### 7.2 Future Enhancement

Even though Hadoop Distributed File System is fault tolerant, scalable and suitable for data intensive applications in this project I have evaluated the read and write performance of HDFS. HDFS follows **single-writer multiple-reader model** that means while write process is in progress, if any client wants to read the same file is not possible in the existing HDFS hence the future work is mostly to propose a model which supports concurrent file access.

## References

[1] Konstantin Shvachko, Hairong kuand, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System" In MSST '10 Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies(MSST), IEEE Computer Society Washingtin, DC, USA 2010, pages 1-10

[2] S. Ghemawat, H. Gobioff, S. Leung. "The Google File System",In Proc.of ACM Symposium on Operating Systems Principles,Lake George ,NY,Oct 2003,pp 29-43

[3] P.H.Carms, W.B Ligon III, R.B.Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters" Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, October 2000, pp. 317-327

[4] Russel Sandber, "The Sun Network File System: Design, Implementation and Experience", Sun Mircrosystems, Inc. 2550 Garcia Sve. Mountain View,CA. 94049(415) 960-7293.

[5] J.J Kistler and M. Satyanarayanan, Disconnected Operation in the Coda File System, Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles, October 13-16, 1991, pages 213-225

[6] John H Howard, An overview of the Andrew File System, Information Technology Center, Carnegie Mellon University

[7] Tom White, "Hadoop: The Definitive Guide", Third Edition, O'reily/Yahoo! Press

[8] Eric Sammer, Hadoop Operations, September 2012: First Edition., O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol,CA 95472.

[9] Apache Hadoop. http://hadoop.apache.org/

[10] P.H.Carms, W.B Ligon III, R.B.Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters," in Proc. Of 4th Annual Linux Showcase and Conference, 2000, pp 317-327

[11] J.Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," In Pric. Of the 6th Se Pig Experience," Symposium on Operating System Design and Implementation, San Francisco CA, Dec. 2004.

[12] A. Gates, O. Natkovich, S. Chopra, P. Kamath, S. Narayanam, C. Olston, B.Reed, S. Srinivasan, U. Srivastava. "Building a High-Level Dataflow System on top of MapReduce: The Pig Experience," In Pric. Of Very Large Data Bases, viol 2 no.2,2008 pp 1414-1425.

[13] S.Ghemawat, H.Gobioff,S.Leung, "The Google file System," In Proc. Of ACM Symposium on Operating Systems Principles, Lake George, NY, Oct 2003, pp 29-43

[14] F.P. Junqueira, B.C. Reed. "The life and times of a zookeeper" In Proc of the 28th ACM Symposium on Principles of Distributed Computing, Calgary, AB, Canada, August 10-12,2009.

[15] Lustre File System. http://www.lustre.org

[16] M.K. McKusick, S.Quinlan. "GFS: Evolution on Fast-Forward," ACM Queue, vol. 7, no.7, New York, NY. August 2009

[17] O. O'Malley, A.C.Murthy. Hadoop Sorts a Petabyte in 16.25 Hours and a Terabyte in 62 seconds. May 2009. http://developer.yahoo.net/blogs/hadoop/2009/05/hadoop_sorts_a_petabyte_in16_2.html

[18] R. Pike, D. Presotto, k.Thompson, H.Tricky, P.Winterbottom, "Use of Name Spaces in Plan9," Operating Systems Review,27(2), April 1993,pages 72-76

[19] S. Radia, "Naming Policies in the spring system," In Proc. Of 1st IEEE workshop on Services in Distributed and Networked Environments, June 1994, pp 164-171.

[20] S. Radia, J. Pachl, "The Peer-Process View of Naming and Remote Execution," IEEE parallel and Distributed Technology, vol.1, no.3, August 1993, pp 71-80.

[21] K.v. Shvachko,"HDFS Scalability: The limits to growth,"; login:. April 2010, pp6-16.

[22] W. Tantisiriroj, S. Patil, G. Gibson. "Data-intensive file systems for Internet services: A rose by any other name…." Technical Report CMUPDL-08-114, Parallel Data Laboratory, Varnegie Mellon University, Pittsburgh, A, October 2008.

[23] A.T Husoo, J.S. Sarma, N. Jain, Z.Shao, P.Chakka,S.Anthony, H.Liu,P.Wyckofff, R.Murthy, "Hive-A Warehousing solution Over a MapReduce Framework," In Proc. Of Very Large Data Bases, vol.2 no.2, August 2009, pp 1626-1629.

## Author Profile

**Linthala Srinithya** received Bachelor of Engineering in Computer Science and Engineering from TRRCE, JNTUH. She is pursuing Master of Technology in Computer Science. Her research interests are Big Data and Analytics, Operating Systems, Data mining, Networking, Web Technologies, Image Processing, Computer Graphics.

**G. Venkata Rami Reddy** has completed his Master of Technology in Computer Science from School Of IT, JNTU, Kukatpally Hyderabad.. He is the Associate Professor and course coordinator of Software Engineering for School of IT, JNTUH. His subjects of interests are Image Processing, Computer Networks, Analysis of Algorithms, Data mining, Operating Systems and Web technologies.