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

Web User Interface of Hadoop Distributed File System

D Dhananjay[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 Distributed file systems forms the basis of those computing paradigms which implement network intensive frameworks. Among various available Distributed File Systems (DFS) Apache Hadoop implemented its own file system known as Hadoop Distributed File System (HDFS) which forms the basis of Hadoop. Hadoop Distributed File System (HDFS) is a distributed file system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster. The main objective of this project is to make Hadoop Distribute File***

Keywords: HDFS, SaaS, PaaS, IaaS, fault-tolerant.

1. Introduction

1.1 Introduction

Apache Hadoop is an open-source software framework for storage and large scale processing of data-sets on clusters of commodity hardware. Hadoop is an Apache top-level project being built and used by a global community of contributors and users. It is licensed under the Apache License 2.0. The Apache Hadoop Framework is composed of the following modules:
Hadoop Common-contains libraries and utilities needed by other Hadoop modules.

1.1.1 Hadoop Distributed File System (HDFS)

A distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.

1.1.2 Hadoop YARN

A resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users applications.

1.1.3 HadoopMapReduce

A programming model for large scale data processing. All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework.

Apache HadoopMapReduce and HDFS components originally derived respectively from Google's MapReduce and Google File System (GFS) papers.

Beyond HDFS, YARN and MapReduce, the entire Apache Hadoop "platform" is now commonly considered to consist of a number of related projects as well Apache Pig, Apache Hive, Apache HBase, Apache Spark, and others. For the end-users, though MapRduce java code is common, any programming language can be used with "Hadoop Streaming" to implement the "map" and "reduce" parts of the user's program. Apache Pig, Apache Hive, Apache Spark among other related projects expose higher level user interfaces like Pig Latin and a SQL variant respectively. The Hadoop framework itself is mostly written in the Java programming language, with some native code in C and command line utilities written as shell-scripts.

1.2Architecture

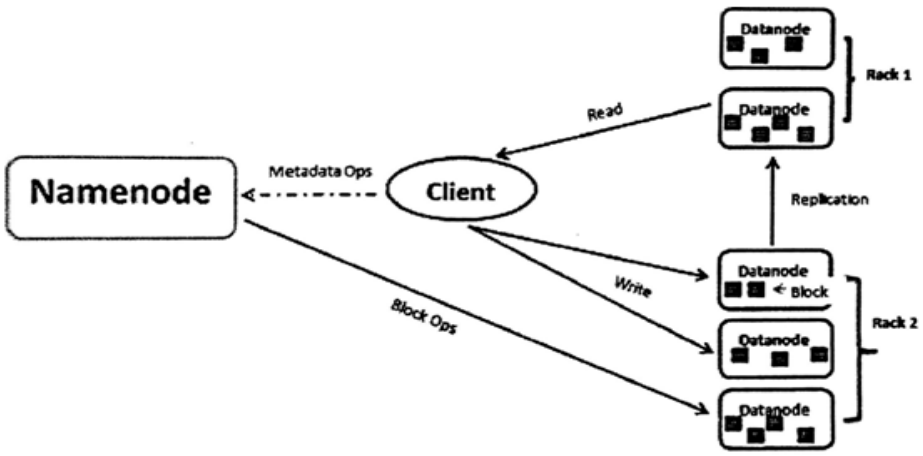


Figure 1.1: HDFS Cluster

A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a Job Tracker, Name Node and Data Node. A Slave or Worker node acts as both a Data Node and Task Tracker, though it is possible to have data-only worker nodes and compute only worker nodes. These are normally used only in nonstandard applications.

Hadoop requires Java Runtime Environment (JRE) 1.6 or higher. The standard start-up and shutdown scripts require Secure Shell (ssh) to be set up between nodes in the cluster. In larger cluster, the HDFS is managed through a dedicated Name Node Server to Host the File system index, and a secondary Name Node that can generate snapshots of the name node's memory structures, thus preventing file-system corruption and reducing loss of data. Similarly, a standalone Job Tracker server can manage job scheduling.

In clusters where the HadoopMapReduce engine is deployed against an alternate file system, the Name Node, secondary Name Node and Data Node architecture of HDFS is replaced by the file-system-specific equivalent.

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.

1.3 Cloud Computing

A cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreement established through negotiation between the service provider and consumers.

Cloud Architecture and Classification

With respect to the ways cloud can be used, the 'de facto' consensus achieved led to the definition of 3 major exploitation levels:

- Infrastructure as a Service (IaaS)
- Platform as a Service (Paas)
- Software as a Service (Saas)

1.4 Motivation

Hadoop Distributed File System implementation is a typical task keeping in view of the large number of clients, large volume of dataset and large size of files. The main objective of this project is to make Hadoop Distributed File System easy for user. To achieve this goal I prepare a Web User Interface by which anyone can use Hadoop easily. I had implemented all the fs shell commands in it.

1.5 Problem Definition

Hadoop Distributed File System does not has any user supported Web GUI Interface. So I have tried to develop a user friendly web application. For that I had implemented all the fs shell commands in it.

1.6 Objective of Project

The main objective of the project is to make Hadoop Distributed File System User friendly, So that anyone can use it easily.

1.7 Limitations of Project

We have tested this project in pseudo-distribution mode. This cannot be implemented in fully distributed mode as there is no provision for IP address to be recognized.

2. Analysis

2.1 Introduction

Hadoop Distributed File System implementation is a typical task keeping in view of the large number of clients, large volume of dataset and large size of files. The main objective behind this project is to make Hadoop Distributed File System easy for user. So that anyone can use HDFS easily. To achieve this goal I prepare a web user interface. I had implemented all the fs shell commands in it.

2.2 FS shell commands

The FileSystem (FS) shell is invoked by bin/hadoop fs <args>. All the FS shell commands take path URIs as arguments. The URI format isscheme://autority/path. For HDFS the scheme is hdfs, and for the local filesystem the scheme is file. The scheme and authority are optional. If not specified, the default scheme specified in the configuration is used. An HDFS file or directory such as /parent/child can be specified ashdfs://namenodehost/parent/child or simply as /parent/child (given that your configuration is set to point to hdfs://namenodehost). Most of the commands in FS shell behave like corresponding Unix commands. Differences are described with each of the commands. Error information is sent to stderr and the output is sent to stdout.

2.2.1 cat

Usage: hadoop fs -cat URI [URI …]
Copies source paths to stdout.

Example:

- Hadoop fs -cat hdfs://nn1.example.com/file1 hdfs://nn2.example.com/file2
- hadoop fs -cat /file3 /user/hadoop/file4

ExitCode:
Returns 0 on success and -1 on error.

2.2.2 chgrp

Usage: hadoop fs -chgrp [-R] GROUP URI [URI …]
Change group association of files. With -R, make the change recursively through the directory structure. The user must be

the owner of files, or else a super-user. Additional information is in the Permissions User Guide.

2.2.3 chmod
Usage: hadoop fs -chmod [-R] <MODE[,MODE]... | OCTALMODE> URI [URI …]
Change the permissions of files. With -R, make the change recursively through the directory structure. The user must be the owner of the file, or else a super-user. Additional information is in the Permissions User Guide.

2.2.4 chown
Usage: hadoop fs -chown [-R] [OWNER][:[GROUP]] URI [URI]
Change the owner of files. With -R, make the change recursively through the directory structure. The user must be a super-user.

2.2.5 copyFromLocal
Usage: hadoop fs -copyFromLocal <localsrc> URI
Similar to put command, except that the source is restricted to a local file reference.

2.2.6 copyToLocal
Usage: hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>
Similar to get command, except that the destination is restricted to a local file reference.

2.2.7 cp
Usage: hadoop fs -cp URI [URI …] <dest>
Copy files from source to destination. This command allows multiple sources as well in which case the destination must be a directory.

2.2.8 du
Usage: hadoop fs -du URI [URI …]
Displays aggregate length of files contained in the directory or the length of a file in case its just a file.

2.2.9 get
Usage: hadoop fs -get [-ignorecrc] [-crc] <src><localdst>
Copy files to the local file system. Files that fail the CRC check may be copied with the -ignorecrc option. Files and CRCs may be copied using the -crcoption.

Example:
- hadoop fs -get /user/hadoop/file localfile
- hadoop fs -get hdfs://nn.example.com/user/hadoop/file localfile

2.2.10 getmerge
Usage: hadoop fs -getmerge <src><localdst> [addnl]
Takes a source directory and a destination file as input and concatenates files in src into the destination local file. Optionally addnl can be set to enable adding a newline character at the end of each file.

2.2.11 ls
Usage: hadoop fs -ls <args>
For a file returns stat on the file with the following format: filename <number of replicas> filesize modification_date modification_time permissionsuserid...groupid

2.2.12 lsr
Usage:hadoop fs- lsr<args>
Recursive version of ls. Similar to Unix ls -R.

2.2.13 mkdir
Usage: hadoop fs -mkdir <paths>
Takes path uri's as argument and creates directories. The behavior is much like unix mkdir -p creating parent directories along the path.

2.2.14 movefromLocal
Usage: dfs -moveFromLocal <src><dst>
Displays a "not implemented" message.

2.2.15 mv
Usage: hadoop fs -mv URI [URI …] <dest>
Moves files from source to destination. This command allows multiple sources as well in which case the destination needs to be a directory. Moving files across filesystems is not permitted.

2.2.16 put
Usage: hadoop fs -put <localsrc> ... <dst>
Copy single src, or multiple srcs from local file system to the destination filesystem. Also reads input from stdin and writes to destination filesystem.

2.2.17 rm
Usage: hadoop fs -rm URI [URI …]
Delete files specified as args. Only deletes non empty directory and files. Refer to rmr for recursive deletes.

2.2.18 rmr
Usage: hadoop fs -rmr URI [URI …]
Recursive version of delete.

2.2.19 setrep
Usage: hadoop fs -setrep [-R] <path>
Changes the replication factor of a file. -R option is for recursively increasing the replication factor of files within a directory.

2.2.20 stat
Usage: hadoop fs -stat URI [URI …]
Returns the stat information on the path.

2.2.21 tail
Usage: hadoop fs -tail [-f] URI
Displays last kilobyte of the file to stdout. -f option can be used as in Unix.

2.2.22 test
Usage: hadoop fs -test -[ezd] URI
Options: -e check to see if the file exists. Return 0 if true.
-z check to see if the file is zero length. Return 0 if true.
-d check return 1 if the path is directory else return 0.

2.2.23 text
Usage: hadoop fs -text <src>

Takes a source file and outputs the file in text format. The allowed formats are zip and TextRecordInputStream.

2.2.24 touchz
Usage: hadoop fs -touchz URI [URI …]
Create a file of zero length.

2.3 Software Requirement Specification

2.3.1 Purpose
In order to uses HDFS in cloud we must be made it user friendly. So that anyone can use it without any hurdle

2.3.2 Scope
The Hadoop Distributed File System (DFS) is a fault tolerant scalable distributed storage component of the Hadoop distributed high perforamance computing platform. Hadoop is asoftware framework that supports largescale distributed data analysis on commodity servers. Hadoop leads cloud service provider to bring the flexibility, agility and massive scalability of the cloud to user, HDFS appears as a traditional file system. You can perform CURD action on files with certain directory path. But, due to the characteristics of distributed storage, there are “NameNode” and “DataNode” which take each of their responsibility. The NameNode is the master of the DataNodes. It provides metadata service within HDFS. The metadata indicates the file mapping of the DataNode. It also accepts operation commands and determines which DataNode should perform the action and replication. The DataNode serves as storage blocks for HDFS.they also respond to command that create, delete, and replication blocks received from the NameNode. In simple term, Hadoop can be used to distributed cloud computing services i.e. it is a distributed platforms. The main objective of this project is to deploy all FS Shell Commands and create a Web GUI so that anyone can use hadoop easily.

2.4 Requirements

Here we dealt with the software and hardware requirements of the project to be made.
Hardware requirements:
- Desktop or laptop
- 2 GB RAM
- 250 GB Hard Disk

Software Requirements:
- Linux
- Java 1.6
- NetBeans IDE
- Hadoop 1.0.4
- OpenSSH

2.5 Software Design Specification

The overall schema of the project include the following aspects.
Data Read: This includes to read all file stored on HDFS. User download the files.
Data Write: The file that is to store the files or data on HDFS. User can also delete the files.

2.6User documentation

A comprehensive user manual is provided in Annexure-I of this documentation.

2.6.1 Functional Requirement
- File should be download according to user request
- File should be upload in HDFS
- We should able to read all files in HDFS
- File requested to read should be granted to read
- File should be remove according to the user request
- We can maker or delete directory and also we can store files in it

2.6.2Non-functional Requirements
Non- functional requirements includes the following
- Performance of hadoop should not degrade.
- The cluster should be scalable
- Reliability of the cluster should be maintained
- Quality of software should improve

3. Design

3.1 Introduction

Hadoop comes with a distributed file system called HDFS, which stands for Hadoop Distributed File System. HDFS is hadoop flagship file system and is the focus of this chapter,butHadoop actually has a general purpose file system abstraction, so we’ll see along the way hadoop integrates with storage system (such as the local file system and Amazon S3).

3.2 The Design of HDFS

HDFS is a file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware.
Lets examine this in more detail:

3.3 Very large Files

Very large in this context means files that are hundreds of megabytes, gigabytes or terabytes in size. There are Hadoop Clusters running today that store petabytes of data.

3.4 Streaming Data Access

HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. A dataset is typically generated or copied from source, then various analyses are performed on that dataset over time. Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

3.5 Commodity Hardware

Hadoop doesnt require expensive, highly reliable hardware to run on. Its designed to run on clusters of commodity hardware (commonly available hardware available from

multiple vendors3) for which the chance of node failure across the cluster is high, at least for large clusters. HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure.

3.6 Low-Latency Data Access

Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS. Remember, HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency.

3.7 Data Flow

3.7.1 Anatomy of File Read

To get an idea of how data flows between the client interacting with HDFS, the namenode and the datanodes, consider Figure 4.1, which shows the main sequence of events when reading a file. The client opens the file it wishes to read by calling open() on the FileSys-tern object, which for HDFS is an instance of Distributed FileSystem (step 1 in Figure 4.1). Distributed FileSystem calls the namenode, using RPC, to determine the locations of the blocks for the first few blocks in the file (step 2).

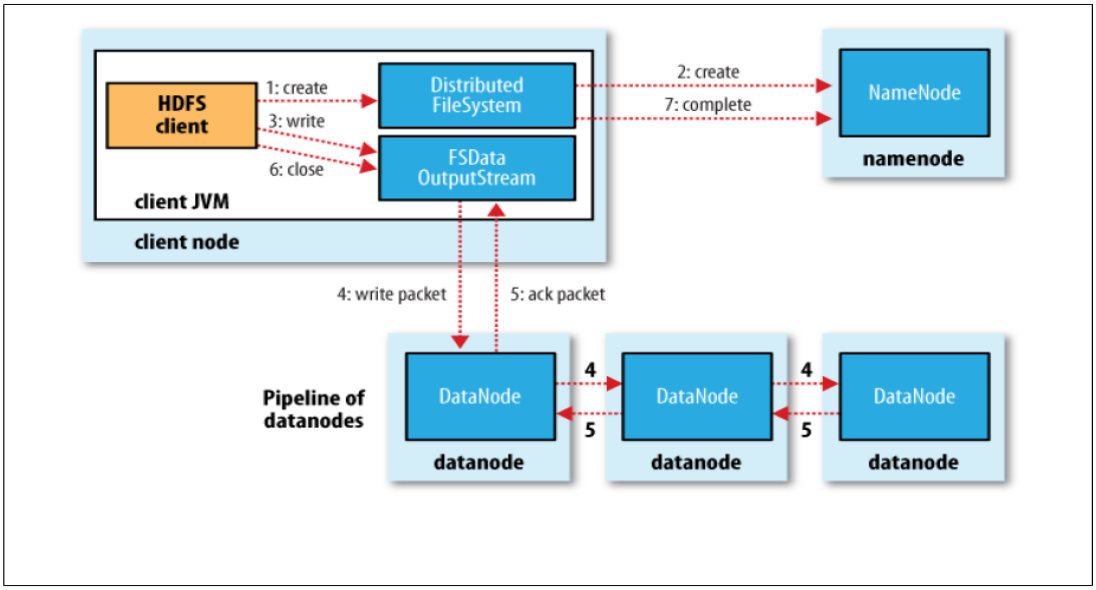


Figure 3.1: A client reading data from HDFS

For each block, the namenode returns the addresses of the datanodes that have a copy of that block. Furthermore, the datanodes are sorted according to their proximity to the client.

If the client is itself a datanode (in the case of a MapReduce task, for instance), then it will read from the local datanode,if it hosts a copy of the block.

The Distributed FileSystem returns an FS DatalnputStream (an input stream that supports file seeks) to the client for it to read data from.

FSDatalnput-Stream in turn wraps a DFSlnputStream, which manages the datanode and namenode I/O.

The client then calls read() on the stream (step 3). DFSInputStream with is reported to the namenode before the DFSInput Stream attempts to read a replica of the block from another datanode.

One important aspect of this design is that the client contacts datanodes directly to retrieve data and is guided by the namenode to the best datan-ode for each block.

This design allows HDFS to scale to a large number of concurrent clients, since the data traffic is spread across all the datanodes in the cluster.

The namenode meanwhile merely has to service block location requests (which it stores in memory, making them very efficient) and does not, for example, serve data, which would quickly become a bottleneck as the number of clients grew.

3.7.2 Anatomy of File Write

Next well look at how files are written to HDFS. Although quite detailed, it is instructive to understand the data flow since it clarifies HDFSs coherency model.

The case was going to consider is the case of creating a new file, writing data to it, then closing the file.

The client creates the file by calling create() on Distributed FileSystem (step 1 in Figure 4.2). DistributedFileSystem makes an RPC call to the namenode to create a new file in the filesystems namespace, with no blocks associated with it (step 2).

The namenode performs various checks to make sure the file doesnt already exist, and that the client has the right

permissions to create the file. If these checks pass, the namenode makes a record of the new file; otherwise, file creation fails and the client is thrown an I0Exception.

The Distributed FileSystem returns an FSDataOutputStream for the client to start writing data to.

Just as in the read case, FSDataOutputStream wraps a DFSOutput Stream, which handles communication with the datanodes and namenode.

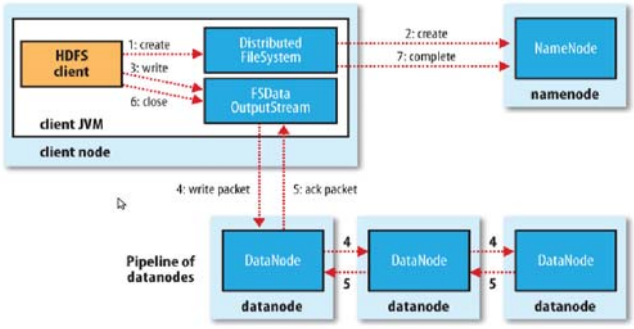


Figure 3.2: A client writing data to HDFS

As the client writes data (step 3), DFSOutputStream splits it into packets,which it writes to an internal queue, called the data queue.

The data queue is consumed by the DataStreamer, whose responsibility it is to ask the namen-ode to allocate new blocks by picking a list of suitable datanodes to store the later on.

The failed datanode is removed from the pipeline and the remainder of the blocks data is written to the two good datanodes in the pipeline. The namenode notices that the block is under-replicated, and it arranges for a further replica to be created on another node. Subsequent blocks are then treated as normal.

Its possible, but unlikely, that multiple datanodes fail while a block is being written. As long as dfs.replication.min replicas (default one) are written, the write will succeed, and the block will be asynchronously replicated across the cluster until its target replication factor is reached (dfs.replication, which de- faults to three).

When the client has finished writing data, it calls close() on the stream (step 6). This action flushes all the remaining packets to the datanode pipeline and waits for acknowledgments before contacting the namenode to signal that the file is complete (step 7). The namenode already knows which blocks the file is made up of (via Data Streamer asking for block allocations), so it only has to wait for blocks to be minimally replicated before returning successfully.

3.8 Conclusion on Design

In this chapter we discuss about the design of Hadoop Distributed File System and its various components. We also study about the anatomy of file reading and file writing

The admin of software in this part can add server to each client with subpart 1, and can delete server with subpart 2, and turn on of the virtual os to each user from subpart 3, and turn off of the virtual os form each client of the system from subpart 4, and at last the admin can list all users vps (Visual Personal Server) information from subpart 5.
Here the admin will control all virtual os for users.

Note:
The admin in subpart 1 can specify details of user's virtual os as os type, hard disk drive, ram. Because some users need more hard disk drive or ram for self system.

Description of test case	*This test case is to check whether file is uploading or not*
Pre-condition	File selected or not
iDescription	File uploading ! OK
Actual Result	File uploading ! OK
Pass/Fail Criteria	Pass when lok` else fail

4. Testing and Validation

4.1 Introduction

Software testingE183 is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation.

Test techniques include, but are not limited to the process of executing a program or application with the intent of finding software bugs (errors or other defects).Software testing can be stated as the process of validating and verifying that a computer program/application/product: meets the requirements that guided its design and development, can be implemented with the same characteristics, and satisfies the needs of stakeholders.

Software Testing, depending on the testing method employed can be implemented at any time in the software development process. Traditionally most of the test effort occurs after the requirements have been defined and the coding process has been completed, but in the Agile approaches most of the test effort is on-going. As such, the methodology of the test is governed by the chosen software development methodology

Software testing methods are traditionally divided into white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test case.

1. White-box testing[4] (also known as clear box testing, glass box testing, transparent box testing and structural testing) tests internal structures or workings of a program, as opposed to the functionality exposed to the end-user.

In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to

testing nodes in a circuit, e.g. in-circuit testing (ICT). While white-box testing can be applied at the unit, integration and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation.

The testers are only aware of what the software is supposed to do, not how it does it. Black-box testing methods include equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing and specification-based testing.

4.2 Design of test cases

I have discussed the various test case designs that can be implemented in this project.

4.3 File Upload

The test case design of File Upload is given below.

Table 4.1: Test case of file upload

Description Of test case	This test case is to check whether file is uploading or not
Pre-condition	File selected or not
iDescription	File uploading ! OK
Actual Result	File uploading ! OK
Pass/Fail Criteria	Pass when lok` else fail

4.4 File Download

The test case design of File Download is given below.

Table 4.2: Test case of file download

Description of test case	This test case is to check whether file is downloading or not
Pre-condition	File selected or not
Expected Result	File downloading ! OK
Actual Result	File downloading ! OK
Pass/Fail Criteria	Pass when 'ok' else fail

4.5 File List

The test case design of File List Store in HDFS is given below

Table 4.3: Test case for File list

Description of test case	This test case is to check whether file is generated or not
Pre-condition	File generated or not
Expected Result	File generated ! OK
Actual Result	File generated ! OK
Pass/Fail Criteria	Pass when `ok` else fail

4.6 File Deletion

The test case design of File Deletion from HDFS is given below

Table 4.4: Test case for File Delete

Description of test case	This test case is to check whether tile is deleted or not
Pre-condition	File deleted or not
Expected Result	File deleted ! OK
Actual Result	File deleted ! OK
Pass/Fail Criteria	Pass when `ok` else fail

4.7 Directory Deletion

The test case design of Directory Deletion from HDFS is given below

Table 4.5: Test case for Directory Delete

Description of test case	This test case is to check whether directory is deleted or not
Pre-condition	Directory deleted or not
Expected Result	Directory deleted! OK
Actual Result	Directory deleted! OK
Pass/Fail Criteria	Pass when 'ole else fail

4.8 Validation

Validation is the process of checking that a software system Meets specifications and that it fulfills its intended purpose. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the software development Lifecycle.

Validation checks that the product design satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements. This is done through dynamic testing and other forms of review. Results of validation testing of test case scenarios are:-

Table 4.6: Validation

TestCase ID	Expected Result	Actual Result	Pass/Fail
6.2.1	File Uploading! Ok	OK	Pass
6.2.2	File Downloading! Ok	OK	Pass
6.2.3	File List Generated! OK	OK	Pass
6.2.4	File Deleted! OK	OK	Pass
6.2.5	Directory Deleted! OK	OK	Pass

4.9 Conclusion on Testing

In this chapter various test cases were designed and validated. The project passed the all five test cases designed. The file is being uploading, downloading, deletion, file listing, directory deletion. This project meets all the specification.

5. Conclusion and Future Work

5.1 Conclusion

Hadoop Distributed File System provides efficient storage of large dataset. Developed by Apache and co-developed by Yahoo it has come a long way from experimental implementation to real world scenario. Providing efficient management for data storage and retrieval it is now one of the most widely used distributed file system for unstructured data. It's read and write algorithm are open to changes.

5.2 Future Work

Hadoop is a fast growing open source technology which constantly needs refinements. One such field is its read-write algorithm. Its many other components such as Hive, map reduce needs development. It posses a big challenge to all BIG DATA managers and developers to define more efficient way to manage and store data.

Furthur extension of project, we can develop an application for novice users or business clients for storing, deleting, uploading, modify huge data of any organization in more ease way.

6. Installation prerequisites"

This project is a simulation of hadoop. Here I am giving the pre-requisites for installing hadoop.

1. ubuntu 12.04: This is the most stable linux package from ubuntua can be installed using wubi.exe software which installs ubuntu above windows. Ubuntu can also be installed using virtual box software.
2. linux-kernal 3.9.0.29:It automatically gets installed with ubuntu 12.04.
3. hadoop-1.0.4:This can be downloaded as a tar package from hadoop.apache.org This is the package with the help of which hadoop will be installed on system.
4. openjdk-6-jdk:Java version 6 or more is required to configure hadoop.
5. Now I am going to discuss how to install hadoop and all other softwares related to it.

6.1 Install openjdk-6-jdk

Open terminal and type the following commands
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudoapt-get install openjdk-6-jdk
This will install Java-6 on ubuntu.

6.2 Create a dedicated hadoop user

It is recommended (not necessary) to add a dedicated user for hadoop with
sudo permissions.
sudo adduser hadoop
sudo adduser hadoop sudo
Now move to the folder where hadoop-1.0.4.tar file is kept and extract it
using following command
tar -xvf hadoop-1.0.4.tar
Now login as hadoop user.
su hadoop

6.3 Installing openssh and generating keys

sudo apt-get install openssh-server
sudo apt-get install openssh-client
ssh –keygen
cd /home/hadoop/.ssh
is (list the keys)
ssh-copy-id -i /home/hadoop/.ssh/id rsa.pub localhost

6.4 Create a folder named temp and change permissions

Before creating tmp folder permission for hadoop-1.0.4 folder needs to be changed.
sudo chown -R hadoop hadoop-1.0.4
sudo chmod 777 hadoop-1.0.4
sudo mkdir /home/hadoop tmp
sudo chown -R hadoop /home/hadoop/tmp
sudo chmod 777 /home/hadoop/tmp

6.5 Start configuring hadoop

sudo gedit /home/hadoop/.bashrc
In this file add the following two lines.
export JAVA HOME---/usr/lib/jvm/java-6-openjdk-amd64
export HADOOP HOME= path to hadoop-1.0.4 directory
Change to bin folder of hadoop-1.0.4/bin
sudo gedit hadoop-env.sh
Add the following line in it
export JAVA HOME=/usr/libijvm/java-6-openjdk-amd64
sudo gedit core-site.xml
Add the following lines in it

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xs1" href="configuration.xs1"?>

<!-- Put site-specific property overrides in this file. - - >

<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>/home/hadoop/tmp</value>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:54310</value>
</property>
</configuration>
```

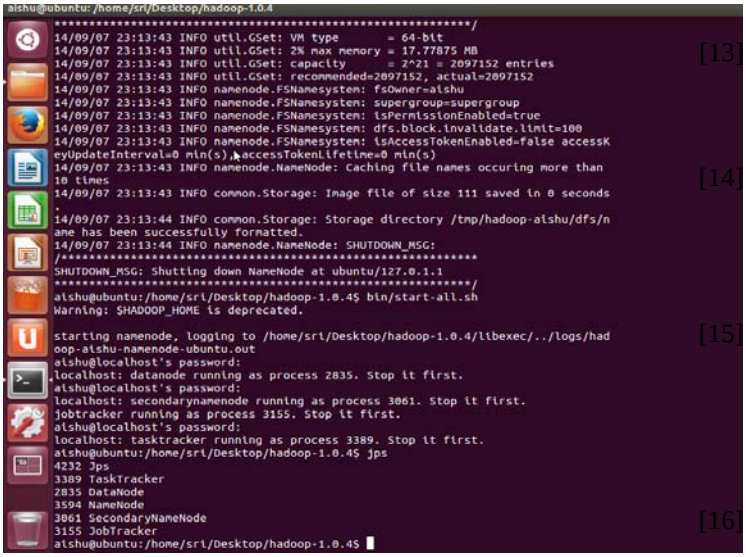
sudo gedit mapred-site.xml
Add the following lines in it

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xs1"
href="configuration.xs1"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>localhost:54311</value>
</property>
</configuration>
```

sudo gedit hdfs-site.xml
Add the following lines in it

```
<?xml version="1.0"?>
<?xml-stylesheet type="textixs1"
href="configuration.xs1"?>.
<!-- Put site-specific property overrides in this file:- -->
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
```

</property>
<property>
</configuration>
Format namenode as
bin/hadoop namenode -format
It will show you a message regarding namenode has been succesfully
formated.



Start the name node by typing **bin/start-all.sh**.
It will start all hadoop daemons.
To check whether all are working type jps.
To stop hadoop type **bin/stop-all.sh.**

References

[1] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler "The Hadoop Distributed File System". Yahoo!,Sunnyvale, California USA,2010,pp.1-10
[2] Apache Hadoop. http://hadoop.apache.org/
[3] S. Ghemawat, H. Gobioff, S. Leung. "The Google filesystem," In Proc. of ACM Symposium on Operating Systems Principles, Lake George,NY, Oct 2003, pp 2943.
[4] IEEE Standard for Software Test Documentation, IEEE Std 829,1998
J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," In Proc. of the 6th Symposium on Operating Systems Design and Implementation, San Francisco CA, Dec. 2004,pp.1-13
Common IO Download http://commons.apache.org/proper/commons-io/download_io.cgi
[5] K. V. Shvachko, "HDFS Scalability: The limits to growth," ;login:.April 2010, pp. 616.
[6] J. Venner, Pro Hadoop. Apress, June 22, 2009.
[7] T. White, Hadoop: The Definitive Guide. O'Reilly Media, Yahoo! Press,June 5, 2009.
[8] InterMezzo.http://www.inter-mezzo.org
[9] Lustre.http://www.lustre.org
[10] Barbara Liskov, Sanjay Ghemawat, Robert Gruber,Paul Johnson, Liuba Shrira, and Michael Williams. Replication in the Harp file system. In 13th Symposium on Operating System Principles, pages 226238, Pacific Grove, CA,October 1991.
[11] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.The Google file system. In 19th Symposium on Operating Systems Principles, pages 29.43, Lake George,New York,2003.
[12] Douglas Thain, Todd Tannenbaum, and Miron Livny.Distributed computing in practice: The Condor experience. Concurrency and Computation: Practice and Experience,2004.
[13] P.H. Carns, 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,200,pp. 317327
[14] Lusture File System. http://www.lustre.org
Software Testing.http://en.wikipedia.org/wiki/Software_testing
Creating Tables with Latex www1.maths.leeds.ac.uk/latex/TableHelp1.pdf
[15] A. Gates, O.Natkovich, S. Chopra, P.kamath, S.Narayanam, C.Olston, B. Reed, S. Srinivasan, U. Srivastava. "Building a Higjh-LevelDataFlow System on top of MapReduce : The Pig Experience," In Proc. Of Very Large Data Bases, vol 2 no. 2, 2009, PP. 14141425 .
[16] S. Radia, "Naming Policies in the spring system," In Proc. Of 1st IEEE Workshop on Services in Distributed and Networked Environments , June 1994, PP. 164171.

Author Profile

Dubellam Dhananjay received Bachelor of Engineering in Computer Science and Engineering from TRRCE, JNTUH. He is pursuing Master of Technology in Computer Science. His 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.