ISSN:0975-9646

The Hadoop distributed file system

Pooja S.Honnutagi

Computer Science & Engineering
VTU Regional Centre Gulbaga,Karnataka,India

Abstract: The flood of data generated from many sources daily. Maintenance of such a data is challenging task. The solution is Hadoop. Hadoop is a framework written in Java for running applications on large clusters of commodity hardware. The Hadoop Distributed File System (HDFS) is designed to be scalable,fault-toleran,distributed storage system that works closely with MapReduce.In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size. Using Hadoop's HDFS and MapReduce programming model we can distribute,process and count *the number of occurrence of each word in large file set.*

Keywords: **Hadoop,HDFS,MapReduse,Namenode, Datanode**

I. INTRODUCTION

Hadoop is a framework written in Java for running applications on large clusters of commodity hardware and incorporates features similar to those of the Google File System (GFS) and of the MapReduce computing paradigm. Hadoop's HDFS is a highly fault-tolerant distributed file system and, like Hadoop in general, designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications that have large data sets. Even if hundreds or thousands of CPU cores are placed on a single machine, it would not be possible to deliver input data to these cores fast enough for processing. Individual hard drives can only sustain read speeds between 60-100 MB/second. These speeds have been increasing over time, but not at the same breakneck pace as processors. Optimistically assuming the upper limit of 100 MB/second, and assuming four independent I/O channels are available to the machine, that provides 400 MB of data every second. A 4 terabyte data set would thus take over 10,000 seconds to read--about three hours just to load the data! With 100 separate machines each with two I/O channels on the job, this drops to three minutes.

Hadoop processes large amount of data by connecting many commodity computers together and making them work in parallel. A theoretical 100-CPU machine would cost a very large amount of money. It would in fact be costlier than 100 single-CPU machines. Hadoop basically ties together smaller and more reasonably priced computers to form a single cost-effective compute cluster. Computation on a large amount of data has been done before in a distributed setting. The simplified programming model is the reason that makes Hadoop unique. In a Hadoop cluster when the data is loaded it is distributed to all the machines of the cluster as shown in Fig 1.

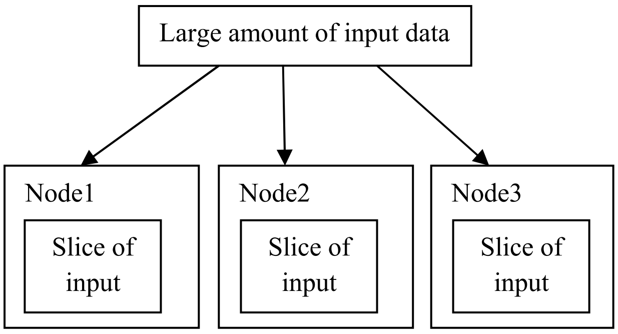


Fig 1 :Data is distributed across nodes at load time

Hadoop DistributedFile System (HDFS) splits the large data files into parts which are managed by different machines in the cluster. Each part is replicated across many machines in a cluster, so that if there is a single machine failure it does not result in data being unavailable. In the Hadoop programming framework data is record oriented. Specific to the application logic, individual input data files are broken into various formats. Subsets of these records are then processed by each process running on a machine in the cluster. Using the knowledge from the DFS these processes are scheduled by the Hadoop framework based on the location of the record or data. The files are spread across the DFS as chunks and are computed by the process running on the node. Hadoop framework helps in preventing unwanted network transfers and strain on network can be obtained by reading data from the local disk directly into the CPU. Thus with hadoop one could have high performance results due to data locality, with their strategy of moving the computation to the data.

II. ARCHITECTURE

A HDFS is filesystem component of Hadoop.HDFS has master/slave architechture.An HDFS cluster consists of single namenode, a master server and many datanodes ,called slaves in the architecture.The HDFS stores filesystem metadata and application data separetly.HDFS stores metadata on separate dedicated server called Namenode and Application data are stored on separate servers called Datanodes. All servers fully connected and communicated with the TCP based protocols.The below fig 2 shows the complete architechture of the HDFS[1].

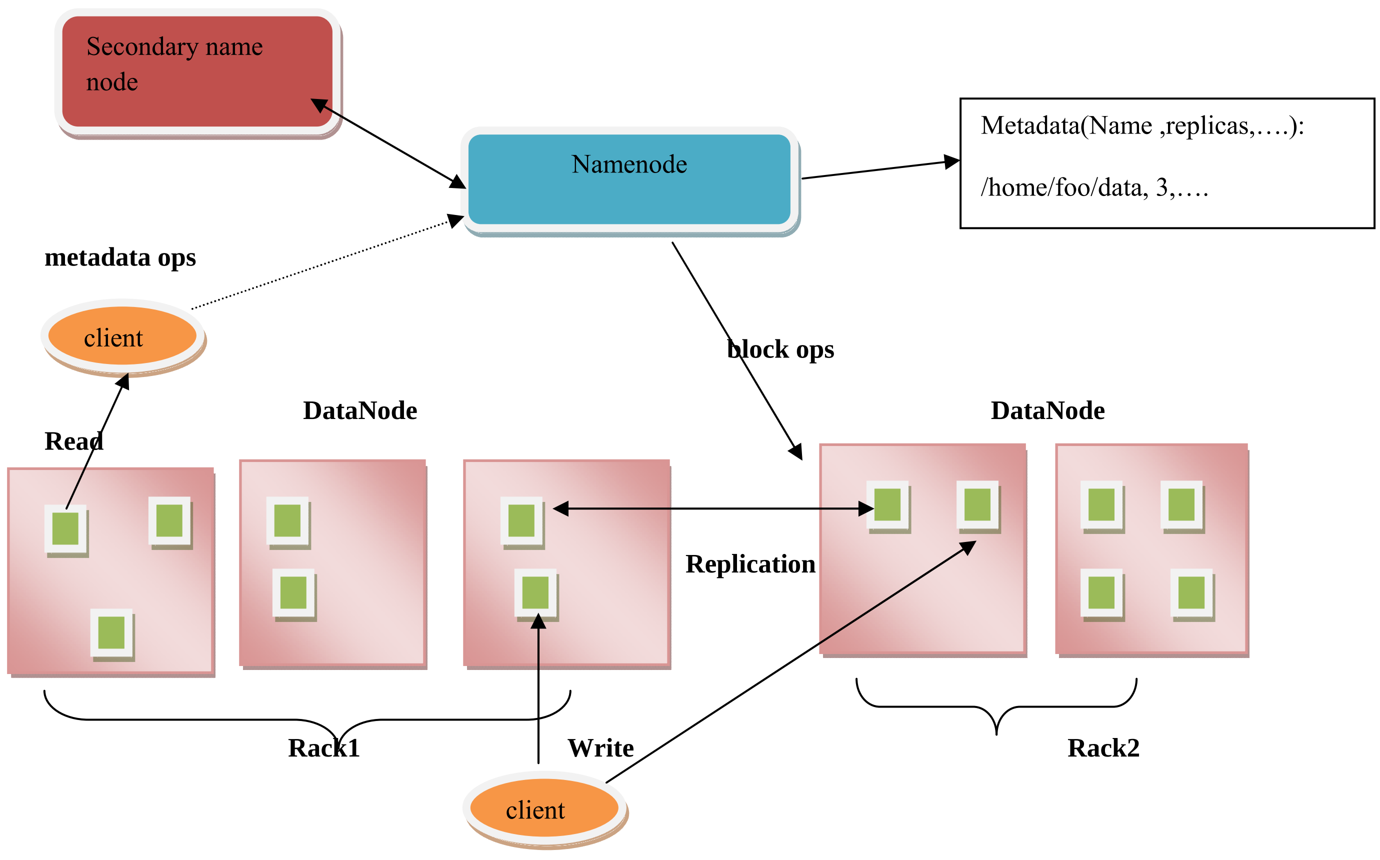


Fig 2:Architecture of HDFS

A. *Namenode*

Namenode holds all the filesystem metadata for the cluster and oversees the health of datanode and coordinates access to data.Namenode is the central controller of the HDFS.It does not hold any cluster data itself.The namenode only knows what blocks make up a file and where those blocks are located in the file.The namenode points clients to the datanodes they needs to talk to and keeps track of the cluster's storage capacity,health of each datanode ,and making sure each block of data is meeting the minimum defiend replica policy.

The namenode maintains file system namespace .Any change to the filesystem namespace or its properties recorded by the namenode.An application can specify the number of replicas of a file that should be maintained by HDFS.Number of copies of the file is called replication factor o that file.This information is stored by the namenode.

The namenode is critical component of the HDFS.without it,clients would not be able to read or write files from HDFS,and it would be impossible to schedule and execute MapReduce jobs.Beacuse of this,its good idea to equip the namenode with a highly redundant enterprise class server configuration;dual server supplies,hot swappable fans,redundant NIC connections etc..

B. *Datanode*

HDFS stores application data in datanode.During startup each datanode connects to the namenode and performs handshake.The purpose of handshake is to verify the namespace id and software version of the datanodes.If either doesnot match that of the namenode and datanode automatically shuts down.

HDFS uses heartbeat messages to detect connectivity between namenode and datanodes.Datanode send heartbeat to the namenode for every three seconds via TCP handshake,using same port number defiend for the namenode daemon,usually TCP 9000.Every tenth heartbeat is the Block report,where the datanode tells the namenode about all the data blocks it has.the block reports allow the Namenode to build its metadata and ensure (3)copies of the blocks exist in different nodes in different racks.

C. *Secondary Namenode*

Hadoop has server role called the Secondary Name Node. A common misconception is that this role provides a high availability backup for the Name Node. This is not the case.The Secondary Name Node occasionally connects to the Name Node (by default, ever hour) and grabs a copy of the Name Node's in-memory metadata and files used to store metadata (both of which may be out of sync). The Secondary Name Node combines this information in a fresh set of files and delivers them back to the Name Node, while keeping a copy for itself.

Should the Name Node die, the files retained by the Secondary Name Node can be used to recover the Name Node. In a busy cluster, the administrator may configure the Secondary Name Node to provide this housekeeping service much more frequently than the default setting of one hour. Maybe every minute

.

D. *HDFS client*

User application access the flie system using HDFS client.Like other file systems,HDFS supports operations to read,write and delete files.When an application reads a file,HDFS client first asks the namenode about list of datanodes that host replicas of the blocks of the file.It then contacts datanodes directly and requests the transfer of the desired block.when client writes,it first asks the namenode to choose datanode to host replicas of the first block of the file.The client organizes the pipeline from node-to-node and sends data.

E. *Data replication*

HDFS replicates file blocks for fault tolerance. An application can specify the number of replicas of a file at the time it is created, and this number can be changed any time after that. The name node makes all decisions concerning block replication. HDFS uses an intelligent replica placement model for reliability and performance. Optimizing replica placement makes HDFS unique from most other distributed file systems, and is facilitated by a rack-aware replica placement policy that uses network bandwidth efficiently.

Large HDFS environments typically operate across multiple installations of computers. Communication between two datanodes in different installations is typically slower than data nodes within the same installation. Therefore, the name node attempts to optimize communications between data nodes. The name node identifies the location of data nodes by their rack IDs.

F. *Rack awareness*

HDFS provides rack awareness.Typically,large clusters are arrange across multiple installations(racks).Network traffic between different nodes within the same installation is more efficient than network traffic across installations.A namenode tries to place replicas of a block on multiple installations for improved fault tolerance.However,HDFS allows administrators to decide on which installation a node belongs.Therefore ,each node knows its rack ID,making it rack aware.

III. File Read And Write Operation On Hdfs

A. *Write operation:*

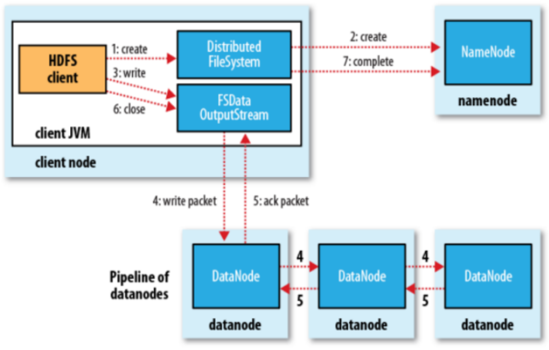


Figure 3:writing data to HDFS

Here we are considering the case that we are going to create a new file, write data to it and will close the file.Now in writing a data to HDFS there are seven steps involved. These seven steps are[2]:

Step 1: The client creates the file by **create()** method on **DistributedFileSystem.**

Step 2: DistributedFileSystem makes an RPC call to the namenode to create a new file in the filesystem's namespace, with no blocks associated with it.The namenode performs various checks to make sure the file doesn't 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 **IOException**. 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.

Step 3: As the client writes data, **DFSOutput Stream** splits it into packets, which it writes to an internal queue, called the **data queue**. The data queue is consumed by the **DataStreamer**, which is responsible for asking the namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas. The list of datanodes forms a pipeline, and here we'll assume the replication level is three, so there are three nodes in the pipeline. The**DataStreamer** streams the packets to the first datanode in the pipeline, which stores the packet and forwards it to the second datanode in the pipeline.

Step 4: Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipeline.

Step 5: DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowledged by datanodes, called the **ack** queue. A packet is removed from the **ack** queue only when it has been acknowledged by all the datanodes in the pipeline.

Step 6: When the client has finished writing data, it calls **close()** on the stream. **Step 7:** 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 The namenode already knows which blocks the file is made up of (via **DataStreamer** asking for block allocations), so it only has to wait for blocks to be minimally replicated before returning successfully.

B. *Read Operation:*

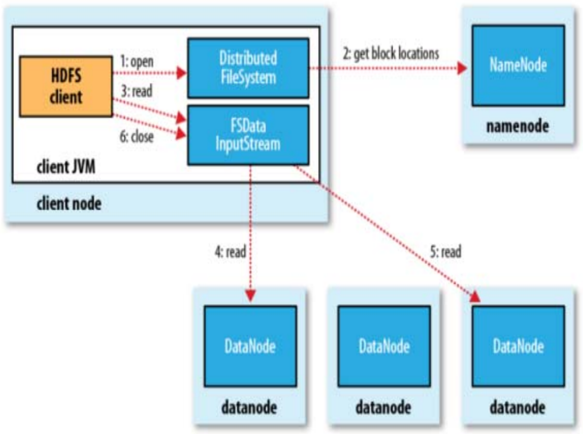


Fig 4:reading the file from HDFS

Fig 4 shows six steps involved in reading the file from HDFS[2]:Let's suppose a **Client** (a HDFS Client) wants to read a file from HDFS. So the steps involved in reading the file is:

Step 1*:* First the Client will open the file by giving a call to open() method on**FileSystem** object, which for HDFS is an instance of **DistributedFileSystem**class.

Step 2*:* DistributedFileSystem calls the **Namenode**, using RPC, to determine the**locations** of the **blocks** for the first few blocks of the file. For each block, the***namenode returns the addresses of all the datanodes*** that have a copy of that block.

The **DistributedFileSystem** returns an object of **FSDataInputStream**(an input stream that supports file seeks) to the client for it to read data from.**FSDataInputStream** in turn wraps a **DFSInputStream**, which manages the datanode and namenode I/O.

Step 3: The client then calls **read()** on the stream. **DFSInputStream**, which has stored the datanode addresses for the first few blocks in the file, then connects to the first **closest** datanode for the first block in the file.

Step 4: Data is streamed from the datanode back to the client, which calls **read()**repeatedly on the stream.

Step 5: When the end of the block is reached, **DFSInputStream** will close the connection to the datanode, then find the best datanode for the next block. This happens transparently to the client, which from its point of view is just reading a continuous stream.

Step 6: Blocks are read in order, with the **DFSInputStream** opening new connections to datanodes as the client reads through the stream. It will also call the namenode to retrieve the datanode locations for the next batch of blocks as needed. When the client has finished reading, it calls **close()** on the FSDataInputStream.

IV. PROCESSING OF DATA DISTRIBUTED ON HDFS

We process data on HDFS parallely using MapReduce programming model.This model is an associated implementation for processing and generating large data sets.user specify a map fuction that process a key/value pair to generate a set of intermediate key/value pair,and a reduce function that merges all intermediate values associated with the same intermediate key.The below figure shows a data flow using MapRedce.

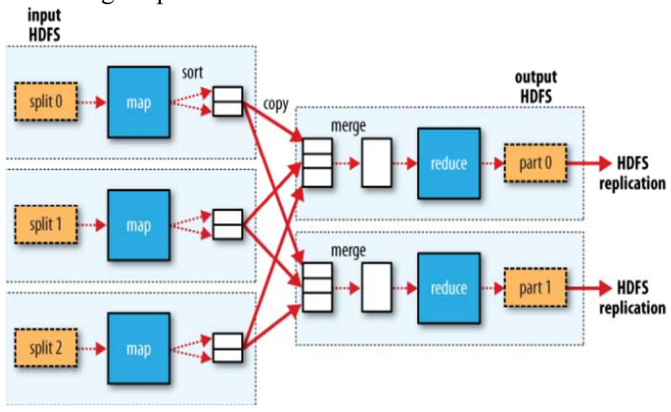


Fig 5: Data Flow in MapReduce

Below steps explains data flow in above fig 5:

Step 1: The system takes input from file system and spilts it up across separate map nodes.

Step 2: The map function or code is run and generates an output for each map node.

Step 3: This output represents a set of intermediate key/value pairs that are moved to reduces nodes as input.

Step 4: The reduce function or code is run and generates an output for each reduce node.

Step 5: The system takes a outputs from each node to aggregate a final view.

A. *WordCount example:*

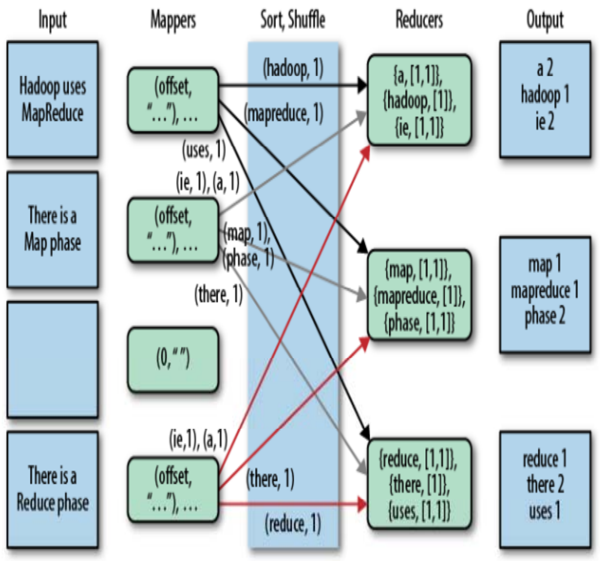


Fig 6:wordcount example

Each Input box on the left-hand side of fig 6 is a separate document. Here are four documents, the third of which is empty and the others contain just a few words, to keep things simple.

By default, a separate Mapper process is invoked to process each document. In real scenarios, large documents might be split and each split would be sent to a separate Mapper. Also, there are techniques for combining many small documents into a single split for a Mapper. The fundamental data structure for input and output in MapReduce is the key-value pair. After each Mapper is started, it is called repeatedly for each line of text from the document. For each call, the key passed to the mapper is the character offset into the document at the start of the line. The corresponding value is the text of the line.

In Word Count, the character offset (key) is discarded. The value, the line of text, is tokenized into words, using one of several possible techniques. Finally, for each word in the line, the mapper outputs a key-value pair, with the word as the key and the number 1 as the value (i.e., the count of "one occurrence"). Note that the output types of the keys and values are different from the input types. Part of Hadoop's magic is the Sort and Shuffle phase that comes next. Hadoop sorts the keyvalue pairs by key and it "shuffles" all pairs with the same key to the same Reducer. There are several possible techniques that can be used to decide which reducer gets which range of keys.

The inputs to each Reducer are again key-value pairs, but this time, each key will be one of the words found by the mappers and the value will be a collection of all the counts emitted by all the mappers for that word. Note that the type of the key and the type of the value collection elements are the same as the types used in the Mapper's output. That is, the key type is a character string and the value collection element type is an integer. To finish the al l key-value pair consisting of each word and the count for that word.gorithm, all the reducer has to do is add up all the counts in the value collection and write a fina

Algorithm

```
1: class Mapper
2: method Map(docid a; doc d)
3: for all term t ε doc d do
4: Emit(term t; count 1)

1: class Reducer
2: method Reduce(term t; counts [c1; c2; : : :])
3: sum=0
4: for all count c ε counts [c1; c2; : : :] do
5: sum=sum + c
6: Emit(term t; count sum)
```

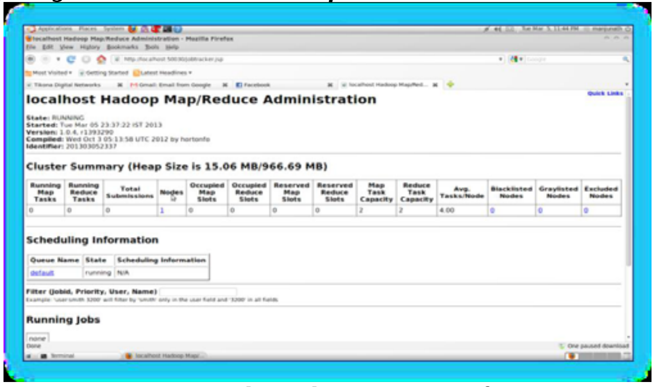
C.**Table 1 shows the Map Reduce program running.**

```
10/04/13 17:43:02 INFO mapred.JobClient: map 0%
reduce 0%
10/04/13 17:43:14 INFO mapred.JobClient: map 66%
reduce 0%
10/04/13 17:43:17 INFO mapred.JobClient: map 100%
reduce 0%
10/04/13 17:43:26 INFO mapred.JobClient: map 100%
reduce 100%
```

Table 1:Map Reduce program running

V. RESULTS

A.The Fig 7 shows the Hadoop Administration interface ,this gives the entire cluster information.



localhost Hadoop Map/Reduce Administration

State: RUNNING
Started: Tue Mar 05 23:37:22 IST 2013
Version: 1.0.4, r1393290
Compiled: Wed Oct 3 05:13:58 UTC 2012 by hortonfo
Identifier: 201303052337

Cluster Summary (Heap Size is 15.06 MB/966.69 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
0	0	0	1	0	0	0	0	2	2	4.00	0	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

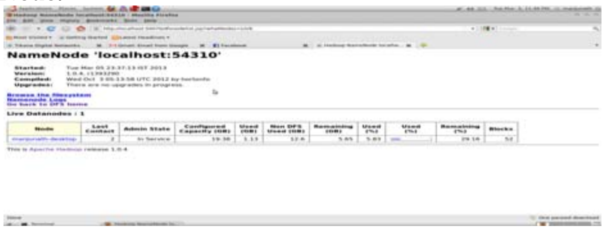
Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

Figure 7: Hadoop Administration Interface

B.The Fig 8 shows the Name Node interface,this gives the total capacity, remaining and used capacity of the Name Node.



NameNode 'localhost:54310'

Started: Tue Mar 05 23:37:13 IST 2013
Version: 1.0.4, r1393290
Compiled: Wed Oct 3 05:13:58 UTC 2012 by hortonfo
Upgrades: There are no upgrades in progress.

Browse the filesystem
Namenode Logs
Go back to DFS home

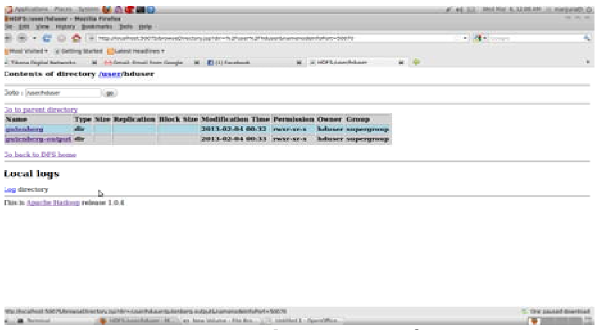
Live Datanodes : 1

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
[illegible]	2	In Service	19.38	1.13	12.6	5.65	5.83	[illegible]	29.16	52

This is Apache Hadoop release 1.0.4

Fig 8: Name Node Interface.

C.The Fig 9 shows the HDFS interface, this gives the information about current data on HDFS.



Contents of directory /user/hduser

Goto : /user/hduser go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
gutenberg	dir				2013-02-04 00:33	rwxr-xr-x	hduser	supergroup
gutenberg-output	dir				2013-02-04 00:33	rwxr-xr-x	hduser	supergroup

Go back to DFS home

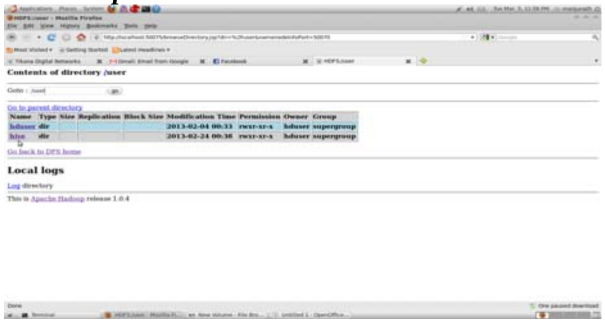
Local logs

Log directory

This is Apache Hadoop release 1.0.4

Fig 9: File System Interface

D.The Fig 10 shows the user accounts on Hadoop along with their permissions



Contents of directory /user

Goto : /user go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
hduser	dir				2013-02-04 00:33	rwxr-xr-x	hduser	supergroup
hive	dir				2013-02-24 00:38	rwxr-xr-x	hduser	supergroup

Go back to DFS home

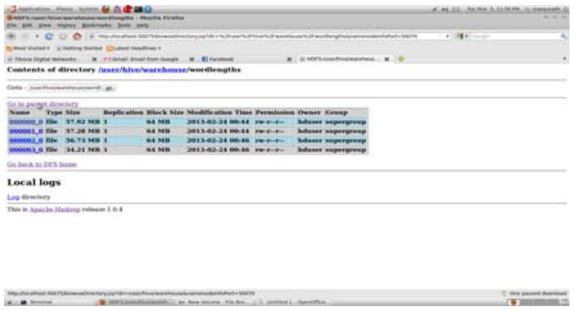
Local logs

Log directory

This is Apache Hadoop release 1.0.4

Fig 10: users on HDFS

E.The Fig 11 shows the Data Chunks distributed over the HDFS.



Contents of directory /user/hive/warehouse/wordlengths

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
000000_0	file	37.92 MB	1	64 MB	2013-02-24 00:44	rw-r--r--	hduser	supergroup
000001_0	file	37.28 MB	1	64 MB	2013-02-24 00:44	rw-r--r--	hduser	supergroup
000002_0	file	36.73 MB	1	64 MB	2013-02-24 00:46	rw-r--r--	hduser	supergroup
000003_0	file	34.21 MB	1	64 MB	2013-02-24 00:46	rw-r--r--	hduser	supergroup

Go back to DFS home

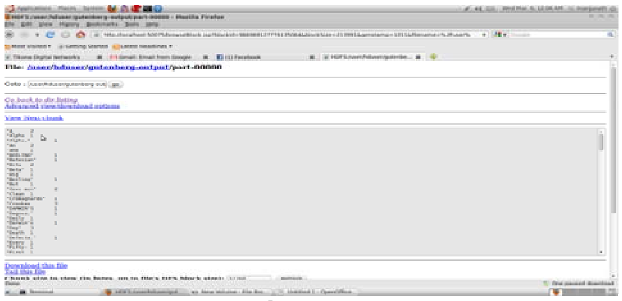
Local logs

Log directory

This is Apache Hadoop release 1.0.4

Fig 11: Processed Data Chunks on HDFS

F.The Fig 12 shows the word count output present in the HDFS.



File: /user/hduser/gutenberg-output/part-00000

Go back to dir listing
Advanced view/download options

View Next chunk

```
[illegible]
```

Download this file
Tail this file

Fig 12: Word Count output on HDFS

VI. CONCLUSION

Hadoop distributed file systems provides a high throughput access to data of an application and is suitable for applications that need to work with large data sets. It is designed to hold terabytes or petabytes of data and provides higher throughput access to this data. Files containing data are stored redundantly across number of machines for higher availability and durability to failure.Moving computation is faster than the moving data. we used MapReduce programming model to process data stored on HDFS.using HDFS we can process and count the number of words in a large data file.

REFERENCES

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

[2] www.bigdataplanet.info

[3] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler,"Hadoop Distributed File System", 2010

[4] Haojun Liao, Jizhong Han, Jinyun Fang "Multi-Dimensional Index on Hadoop Distributed File System", Fifth IEEE International Conference on Networking, Architecture, and Storage ,2010

[5] Kyriacos Talattinis, Aikaterini Sidiropoulou, Konstantinos Chalkias, and George Stephanides, "Parallel Collection of Live Data Using Hadoop", in 14th Panhellenic Conference on Informatics, 2010

[6] Shafer, J, "The Hadoop Distributed File System: Balancing Portability and Performance", 2010

[7] Zhi-Dan Zhao, " User-Based collaborative Filtering Recommendation Algorithms on Hadoop", 2010

[8] Rini T. Kaushik, Milind Bhandarkar, "Evolution and Analysis of GreenHDFS",2010.

[9] Garhan Attebury, Andrew Baranovski, "Hadoop Destributd File System for Grid",2009

[10]K. V. Shvachko, "HDFS Scalability: The limits to growth," ;login:.April 2010,pp. 6–16.

[11]Jeffrey Dean and Sanjay Ghemawat "MapReduce: Simplified Data Processing on Large Clusters"GoogleInc.,