High Performance Fault-Tolerant Hadoop Distributed File System

Yelakala Pragna
Department of Computer Networks and Information Security,
MVGR College of Engineering, Vizianagaram, Andhra Pradesh
pragnabujji555@gmail.com

Abstract – The Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. Huge amounts of data generated from many sources daily. Maintenance of such data is a challenging task. One proposing solution is to use Hadoop. The solution provided by Google, "Doug Cutting" and his team developed an Open Source Project called 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-tolerant, distributed storage system. Hadoop's HDFS is a highly fault-tolerant distributed file system and, like Hadoop in general, designed to be deployed on low-cost hardware. The HDFS stores filesystem Metadata and application data separately. HDFS stores Metadata on separate dedicated server called NameNode and application data stored on separate servers called DataNodes. The file system data is accessed via HDFS clients, which first contact the NameNode data location and then transfer data to (write) or from (read) the specified DataNodes. Download file request chooses only one of the servers to download. Other replicated servers are not used. As the file size increases the download time increases. In this paper we work on three policies for selection of blocks. Those are first, random and loadbased. By observing the results the speed of download time for file is 'first' runs slower than 'random' and 'random' runs slower than 'loadbased'.

Keywords— *Hadoop, HDFS, NameNode, DataNode, Fault Tolerant, Distributed System.*

*****

I. INTRODUCTION

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

Hadoop Distributed File System (HDFS) is a highly scalable, reliable and manageable file system. It supports parallel reading and processing of the data. It also supports read, rename and append operations. It doesn't support random write operations. HDFS is fault tolerant and is redundantly stored by multiple replicas of data kept in the system. It tolerates disk and node failures because of the built in redundancy. The cluster manages addition and removal of nodes automatically without requiring any operational intervention.

II. RELATED WORK

Today Technology is rapidly changes and it is common that every one depends upon the new technologies and uses the server for storing and managing the database. Over the network there are too large set of database and now a day it is challengeable for manage those database. In the age of Big-Data, Hadoop has evolved for handling those dataset. Hadoop is an open source project based on distributed computing having HDFS file system (Hadoop Distributed File System). Hadoop have many advantages that make them highly useful it has fault-tolerance capability and it can be deployed on low cost machines. Hadoop is useful for high volume of data set and it also provides the high speed access to the data set.

The Hadoop framework is used to process the Bigdata applications. It joins multiple datasets together. There are different step in the modelling of the Hadoop framework. First is the storing of the contents into the HDFS. After the contents are stored, we can process the data using the Mapreduce concept. HDFS splits the contents into different chunks and save in different DataNodes of the hadoop cluster.

HDFS gives the programmer unlimited storage and is the only reason behind turning to Hadoop. But when it comes to storing lot of small files there is a big problem. HDFS is capable of handling large files which are GB or TB in size. Hadoop works better with a small number of large files and not with large number of small files. Large number of small files takes up lots of memory on the Namenode. Each small file generates a map task and hence there are too many such map tasks with insufficient input. Storing and transforming small size file in HDFS creates an overhead to map reduce program which greatly affects the performance of Namenode.

Bigdata problems are handled effectively, using the concepts of hadoop. Hadoop is open source software developed by the Apache. It acts as cross platform operating system. Hadoop contains the distributed file system in order to handle the large range of data. Hadoop has many features, like reliability, data locality, cost effectiveness and efficient computation etc.

The term ‘Big Data’ describes innovative techniques and technologies to capture, store, distribute, manage and analyze petabyte- or larger-sized datasets with high-velocity and different structures. Big data can be structured, unstructured or semi-structured, resulting in incapability of conventional data management methods. Data is generated from various different sources and can arrive in the system at various rates. In order to process these large amounts of data in an inexpensive and efficient way, parallelism is used. Big Data is a data whose scale, diversity, and complexity require new architecture, techniques, algorithms, and analytics to manage it and extract value and hidden knowledge from it. Hadoop is the core platform for structuring Big Data, and solves the problem of making it useful for analytics purposes. Hadoop is an open source software project that enables the distributed processing of large data sets across clusters of commodity servers. It is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance.

Big Data Technology helps to store, manage and process high volume and variety of data in cost & time effective manner. It analyzes data in its native form, which could be unstructured, structured or streaming. It captures data from live events in real time. It has a very well defined and strong system failure mechanism which provides high-availability. It handles system uptime and downtime. Using commodity hardware for data storage and analysis. Maintain multiple copies of the same data across clusters. It stores data in blocks in different machines and then merges them on demand.

Big Data is frequently generated and a large volume of data is updated around the clock across the globe by the users. Handling large volume of data in a real time environment is a challenging task. Distributed File System is one of the strategies to handle large volume of data in the real time. Distributed file system is a collection of independent computers that appear to the users of the system as a single coherent system**.** In Distributed file system common files can be shared between the nodes, the drawbacks are scalability, replication, availability and very expensive to buy a hardware server. To overcome this issue Hadoop Distributed File System came into existence. Hadoop distributed file system to run on cluster of commodity hardware like personal computer and laptop. HDFS provides the scalable, fault-tolerance, cost-efficient storage for Bigdata.

HDFS is an excellent choice for supporting big data analysis. The service includes “NameNode” and multiple “DataNodes” running on a commodity hardware cluster. It provides the highest levels of performance, when the whole cluster is in the alike physical rack in the data centre. In Hadoop cluster, data is distributed over the machines of the cluster when it is loaded. The NameNode act as Master, stores Metadata information about actual data location of each block, file names and file properties. The DataNode act as Slave, stores actual data block information. In HDFS files are broken into blocks. The blocks are stored as files on the DataNodes. In HDFS cluster there is a node called NameNode that manages the file system namespace.

III. ARCHITECTURE

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

HDFS has a Master and Slaves architecture in which the master is called the NameNode and slaves are called DataNodes. An HDFS cluster consists of a single NameNode that manages the file system namespace (or Metadata) and controls access to the files by the client applications and multiple DataNodes (in hundreds or thousands) where each DataNode manages file storage and storage device attached to it.

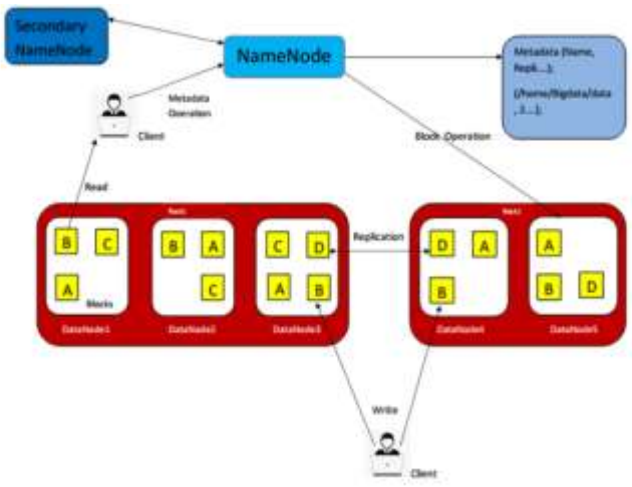


Figure.1 *Architecture of HDFS*

While storing a file, HDFS internally splits it into one or more blocks. These blocks are stored in a set of slaves, called DataNodes; to ensure that parallel writes or reads can be done even on a single file. Multiple copies of each block are stored per replication factor for making the platform fault- tolerant.

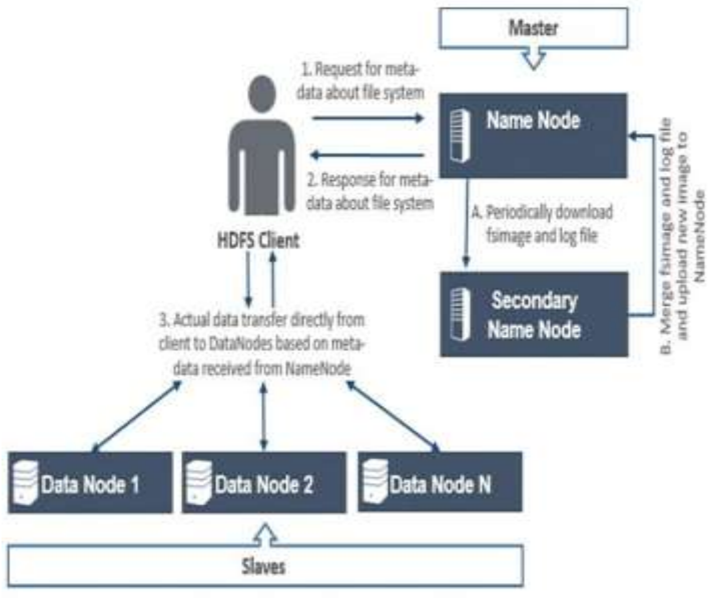

Master
1. Request for meta-data about file system
2. Response for meta-data about file system
Name Node
HDFS Client
A. Periodically download fsimage and log file
B. Merge fsimage and log file and upload new image to NameNode
Secondary Name Node
3. Actual data transfer directly from client to DataNodes based on meta-data received from NameNode
Data Node 1
Data Node 2
Data Node N
Slaves

Figure.2 *How a client reads and writes to and from HDFS*

Reading files in Hadoop Distributed File System, the HDFS Client consults NameNode for Metadata about file system. NameNode response for Metadata about file system. NameNode will return the addresses of all DataNodes. By using read() method HDFS Client connect to the DataNodes identifying a list of DataNodes where a block is hosted and selecting DataNode. Actual data transfer directly from HDFS Client to DataNodes based on Metadata received from DataNode.

NameNode:

NameNode (NN) act as Master, it stores Metadata information about actual data location of each data block, file name and file properties. The NameNode is also responsible for managing file system namespace operations, including opening, closing, and renaming files and directories. The NameNode records any changes to the file system namespace or its properties. The NameNode contains information related to the replication factor of a file, along with the map of the blocks of each individual file to DataNodes where those blocks exist.

HDFS works by dividing large files into smaller pieces called blocks. The blocks are stored in the DataNodes. It is the responsibility of the NameNode to know what blocks on which DataNodes make up a complete file. The NameNode managing all access to the files, including reads, writes, create, deletes, and replication of data blocks on the DataNodes. System namespace is the complete collection of all the files in the cluster. NameNode will control this namespace. NameNode and the strong relationship between DataNodes and they operate in a loosely coupled mode. In a typical configuration, you find one NameNode and possibly one or more DataNodes running on one or more physical servers in the rack. The NameNode is smarter than any DataNode. NameNode is so critical for correct operation of the cluster; it can and should be replicated to guard against a single point failure. HDFS break files into a related collection of little blocks. These blocks are distributed among the DataNodes in the HDFS cluster and are managed by the NameNode. NameNode uses a rack ID to keep track of the DataNodes in the cluster.

DataNode:

DataNode (DN) act as Slave, stores actual data block information. DataNodes are responsible for serving read and write requests from the HDFS clients and perform operations such as block creation, deletion, and replication when the NameNode request them to do so. Store and retrieve blocks when they are told to by the client applications or by the NameNode), and they report back to the NameNode periodically with lists of blocks that they are managing, to keep the NameNode up to date on the current status.

A client application consults the NameNode to get Metadata information about the file system. It directly connects to Datanodes directly so that they can transfer data back and forth between the client and the Datanodes. The client communicates with the NameNode to get only Metadata; the actual data transfer happens between the client and the Datanodes. The NameNode is not involved in the actual data transfer.

Secondary NameNode:

Secondary NameNode keeps update Metadata information and periodically compare with main NameNode information.

HDFS client:

A user application can use the HDFS client to access the file system. HDFS can manage read, write, copy and delete operations.

IV. HDFS FAULT-TOLERANT

Fault tolerance in HDFS refers to the working strength of a system in unfavourable conditions and how that system can handle such situation. HDFS is highly fault tolerant. It handles faults by the process of replica creation. The replica of user's data is created on different machines in the HDFS cluster. So whenever if any machine in the cluster goes down, then data can be accessed from other machine in which same copy of data was created. HDFS also maintains the replication factor by creating replica of data on other available machines in the cluster if suddenly one machine fails. Recovery & Fault tolerance handles the following cases:

- When update on one of the DataNodes fails
- Recovery of a DataNode identifying invalid block versions
- When one DataNode is taken out, to maintain the fault-level the blocks must be copied to new DataNodes

How this feature will be achieved?

HDFS achieves fault tolerance mechanism by replication process. In HDFS whenever a file is stored by the user, then firstly that file is divided into blocks and then these blocks of data are distributed across different machines present in HDFS cluster. After this, replica of each block is created on other machines present in the cluster. By default HDFS creates 3 copies of a file on other machines present in the cluster when replication fails in 2. So due some reason if any machine on the HDFS goes down or fails, then also user can easily access that data from other machines in the cluster in which replica of file is present. Hence HDFS provides faster file read and write mechanism, due to its unique feature of distributed storage.

Example:

Suppose a user's present in a file named 'f'. This data FILE is divided in into blocks say B1, B2, B3 and send to Master. Now master sends these blocks to the slaves say S1, S2, and S3. Now slaves creates replica of these blocks to the other slaves present in the cluster say S4, S5 and S6. Hence multiple copies of blocks are created on slaves. Say S1 contains B1 and B2, S2 contains B2 and B3, S3 contains B3 and B1, S4 contains B2 and B3, S5 contains B3 and B1, S6 contains B1 and B2. Now if due to some reasons slave S4 crashed. Then data present in S4 was B2 and B3 become unavailable. But we don't have to worry because we can get the blocks B2 and B3 from other slave S2. Hence in unfavourable conditions also our data doesn't get lost. Hence HDFS is highly fault tolerant.

V. REPLICATION FACTOR

The replication Factor (n) is a property that can be set in the HDFS configuration file that will allow you to adjust the global replication factor for the entire cluster. For each block stored in HDFS, there will be "n-1" duplicated blocks distributed across the cluster. For example, if the replication factor was set to 3 there would be one original block and two replicas.

Record that each file is broken into multiple data blocks. Now you can explore how these data blocks get stored. By default, each block of a file is stored three times on three different DataNodes: The replication factor configuration property has a default value of 3 but it can be changed.

When a file is created, an application can specify the number of replicas of each block of the file that HDFS must maintain. Multiple copies or replicas of each block make it fault tolerant: If one copy is not accessible or gets corrupted, the data can be read from the other copy. The number of copies of each block is called the replication factor for a file, and it applies to all blocks of a file. An application or job can also specify the number of replicas of a file that HDFS should maintain. The number of copies or replicas of each block of a file is called the replication factor of that file.

The NameNode has the responsibility of ensuring that the number of copies or replicas of each block is maintained according to the applicable replication factor for each file. If necessary, it instructs the appropriate DataNodes to maintain the defined replication factor for each block of a file. Each DataNode in the cluster periodically sends a heartbeat signal and a block-report to the NameNode. When the NameNode receives the heartbeat signal, it implies that the DataNode is active and functioning properly. A block-report from a DataNode contains a list of all blocks on that specific DataNode.

Example:

Replicated servers	Fault-tolerance level
2	1 fail
3	2 fail
4	3 fail etc…

The above table consists of replicated servers and fault-tolerance level. Use block size as complete file size, with fault-tolerance level 2 (the complete file replicated at 3 servers). If one of the servers is fails there is no loss of data because the same copy is replicated to other servers also.

For example, you need to change the replication factor configuration to 1 if you have a single-node cluster. You can even set the replication factor to 2, which requires double the storage space but ensures availability in case a DataNode fails. You can change the replication factor to 4 or higher, which will eventually improve the performance of the read operation at the cost of a more expensive write operation, and with more storage space requirement to store additional copies.

VI. READ OPERATIONS IN HDFS

While reading files in HDFS, the NameNode acts as a Server and the DataNode acts as Clients. Server can read a file in three ways/modes:

- First
- Random
- Loadbased

Case 1: First one in the list is selected as the DataNode that provides the block. The other servers are underutilized, this becomes 'passive replication' (primary-backup) model. Only the first DataNode in the list is consulted to download the file.

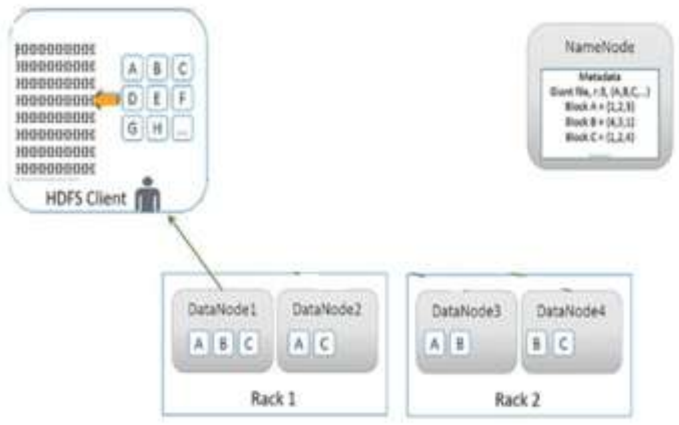


Figure.3 *Using policy as 'first'*

As you can see in Figure, the HDFS Client uses the policy as 'first'. The Rack1 contains the DataNode1 and DataNode2. In that the HDFS Client selects the first DataNode in the list. The DataNode1 doesn't contain all the blocks and at that time the HDFS Client misses the required blocks.

Case 2: Random (better than case 1 as the requests will be distributed). The DataNode is randomly selected from the list of DataNodes that lost the block.

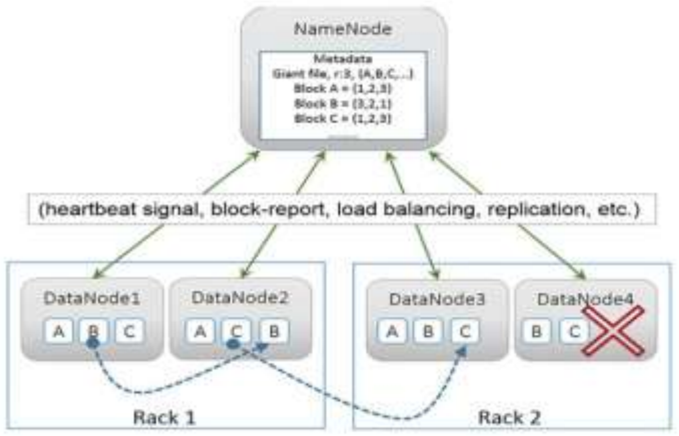


Figure.4 *Using policy as 'random'*

As you can see in Figure, the HDFS Client uses the policy 'random'. 'random' is better than 'first' policy. The figure contains Rack1 and Rack2. The Rack1 contains DataNode1 and DataNode2. The Rack2 contains DataNode3 and DataNode4. The HDFS Client randomly selects the blocks in the DataNodes. Observe thatthe dead DataNode4 contained blocks B and C, so NameNode instructs other DataNodes, in the cluster that contain blocks B and C, to replicate it in such a manner that it is load-balanced and the replication factor is maintained for that specific block. The name node then updates its file system namespace with the latest information regarding blocks and where they exist now. Though the DataNode4 failed yet the HDFS Client doesn't lose any blocks because the same blocks available in other DataNodes.

Case 3: Based on the estimated server load (current queue + response time observed in earlier requests). The 'loadbased' is the better than first two cases. The download time for file is faster than both 'first' and 'random'.

Whereas "first" runs slower than "random" and "random" runs slower than "loadbased".

The blocks are accessed in two ways:

- OnDemand
- Look a block ahead

OnDemand means access which ever block you want. Whereas Look a block ahead means the process is being done in one block at the same time second block is also retrieved.
HDFS works best with small number of very large files for storing large data sets that the applications need. While storing files, HDFS internally splits a file content into one or more data blocks. These data blocks are stored on a set of slaves called DataNodes, to ensure a parallel data read or write.

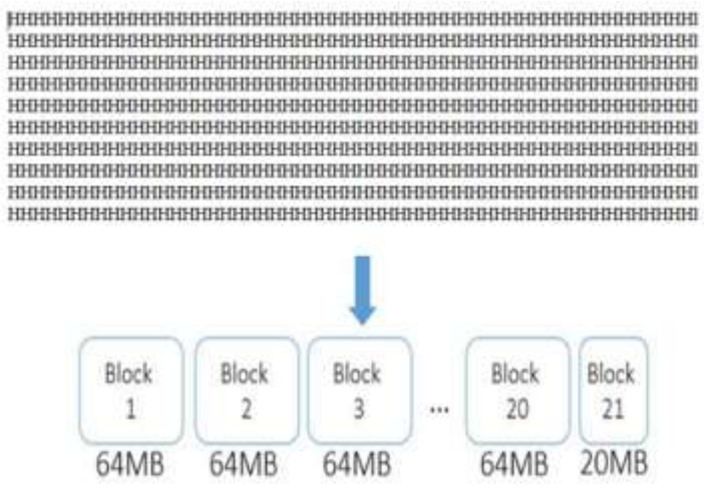


Figure.5 *File split process when reading to HDFS*

All blocks of a file are of the same size except the last block, which can be either of the same size or smaller. HDFS stores each file as a sequence of blocks, with each block stored as a separate file in the local file system.

VII. EVALUATION AND RESULTS

Hadoop Distributed File System (HDFS) was designed to hold and manage large amounts of data; therefore typical HDFS block sizes significantly larger than the block sizes you would see for a traditional file system (for example, the size of the file is 104,857,600 bytes and the size of the block is 5120000 bytes). The block size setting is used by HDFS to divide files into blocks and then distribute those blocks across the cluster. For example, if a cluster is using a block size of 105MB, and a 100MB text file was put in to HDFS,

HDFS would split the file into twenty blocks and distribute the chunks to the DataNodes in the cluster.

Reading files in HDFS, the DataNodes ping the IP address of the NameNode. Based on the size of the block used the NameNode is chopping the blocks in DataNodes. All DataNodes have the same copy of blocks. If any DataNode fails there is no loss of data; whereas the copy of a block is available in other DataNodes. For that reason the HDFS is fault-tolerant.

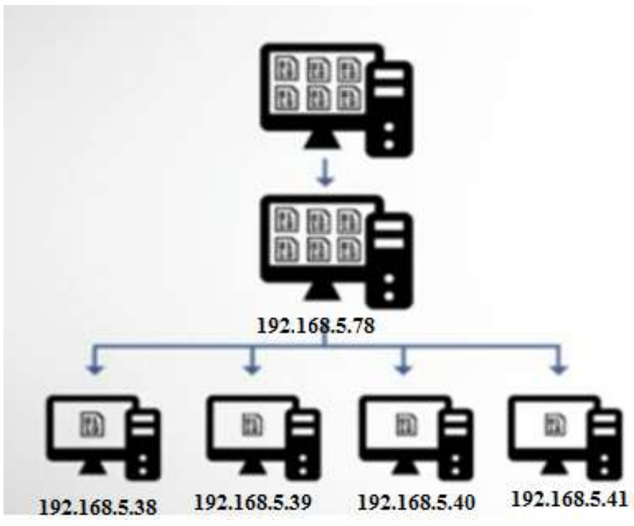
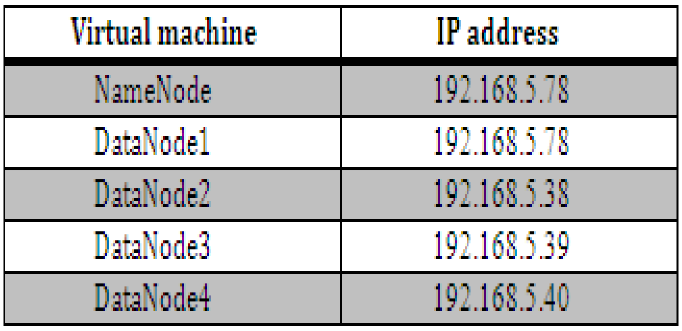


Figure.6 *Cluster topology for experiment*

Reading files in HDFS, the client create “dfs” folder in DataNode. In that, again create the DataNodeMetadata and NameNodeMetadata folders. The DataNodeMetadata again create the DataNode1 on 192.168.5.78, like the DataNode2 on 192.168.5.38, DataNode3 on 192.168.5.39 and DataNode4 on 192.168.5.40 in another command prompt and pass localhost port number at run time. Reading files in HDFS, the cluster topology shows the following table.



Virtual machine	IP address
NameNode	192.168.5.78
DataNode1	192.168.5.78
DataNode2	192.168.5.38
DataNode3	192.168.5.39
DataNode4	192.168.5.40

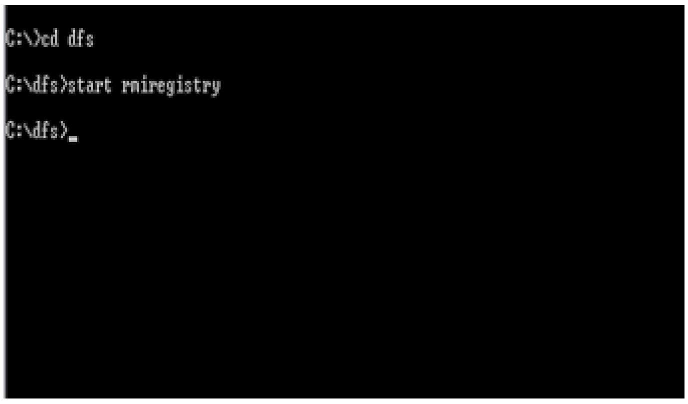
Start the rmiregistry
Start the NameNode on localhost like
start java commands.StartNameNode localhost
On 192.168.5.78
start java commands.StartDataNode DataNode1 localhost
On 192.168.5.38
start java commands.StartDataNode DataNode2 192.168.5.78
On 192.168.5.39
start java commands.StartDataNode DataNode3 192.168.5.78
On 192.168.5.40
start java commands.StartDataNode DataNode4 192.168.5.78

RESULTS:

Steps to run this RMI application:

Save the entire java file into a directory and name it as “dfs”.

start rmiregistry



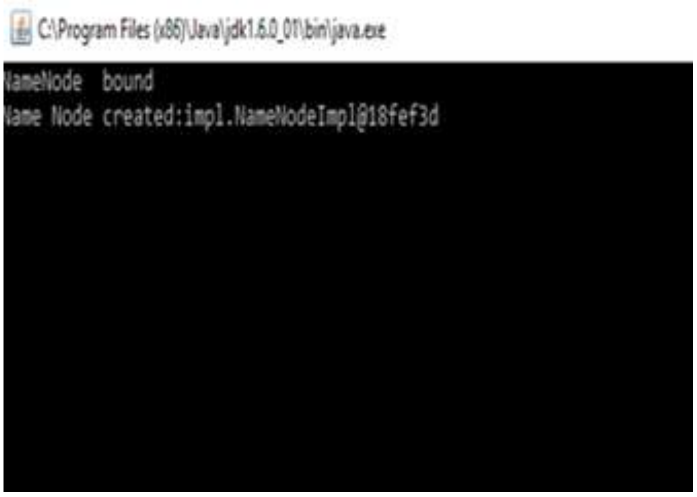
```
C:\>cd dfs

C:\dfs>start rmiregistry

C:\dfs>_
```

Figure7: *Start rmiregistry*

start java commands.StartNameNode localhost



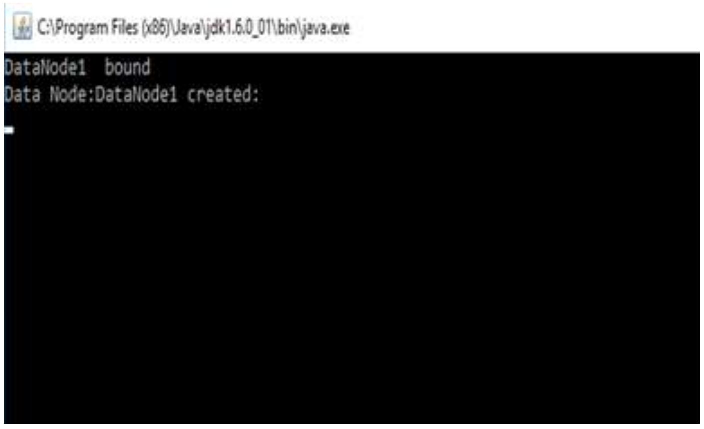
```
C:\Program Files (x86)\Java\jdk1.6.0_01\bin\java.exe
```

```
NameNode  bound
Name Node created:impl.NameNodeImpl@18fef3d
```

Figure8: *Start NameNode*

start java commands.StartDataNode DataNode1 localhost

ISSN: 2321-8169



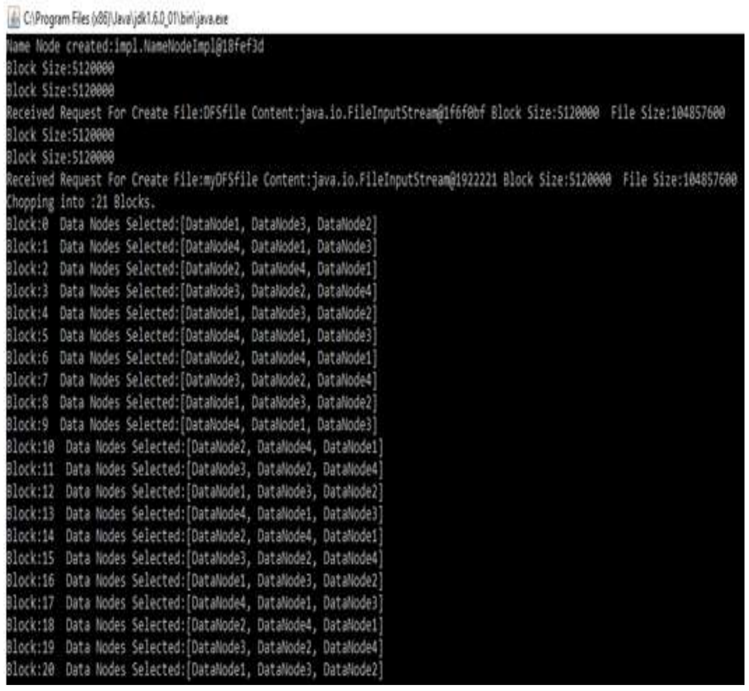
```
C:\Program Files (x86)\Java\jdk1.6.0_01\bin\java.exe
DataNode1  bound
Data Node:DataNode1 created:
```

Figure9: *Start DataNode1*

Like DataNode1 on 192.168.5.78, start the DataNode2 on 192.168.5.38, DataNode3 on 192.168.5.39 and DataNode4 on 192.168.5.40 in another command prompt and pass localhost port number at run time.

The block size setting is used by HDFS to divide files into blocks and then distribute those blocks across the cluster.

For example, if a cluster is using a file size of 100MB, and a block size of 5 MB text file was put in to HDFS, HDFS would split the file based on the file size and block size. By using the following information it divides into twenty blocks and distributes the chunks to the DataNodes in the cluster like this way.



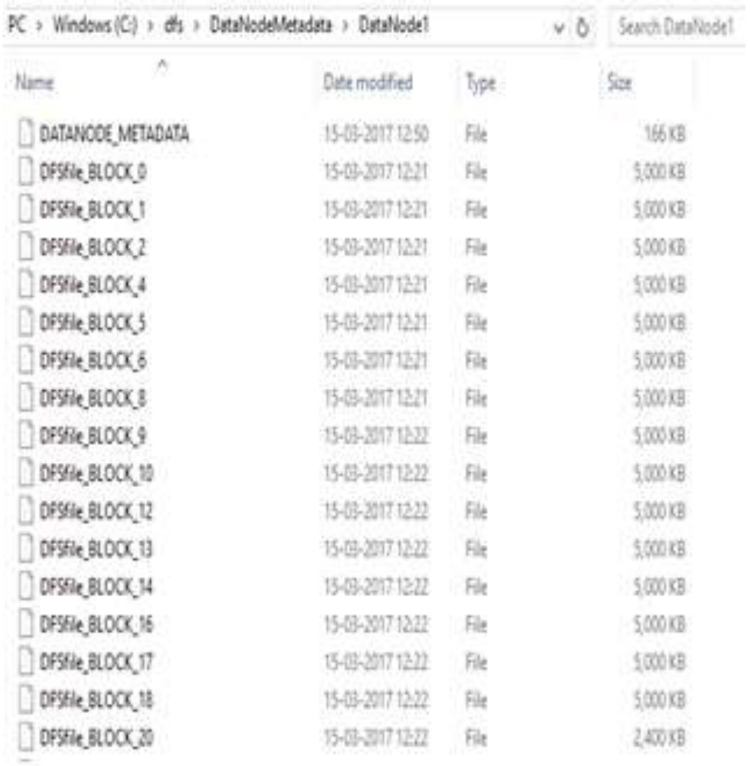
```
C:\Program Files (x86)\Java\jdk1.6.0_01\bin\java.exe
Name Node created:impl.NameNodeImpl@18fef3d
Block Size:5120000
Block Size:5120000
Received Request For Create File:DFSfile Content:java.io.FileInputStream@1f6f0bf Block Size:5120000  File Size:104857600
Block Size:5120000
Block Size:5120000
Received Request For Create File:myDFSfile Content:java.io.FileInputStream@1922221 Block Size:5120000  File Size:104857600
Chopping into :21 Blocks.
Block:0  Data Nodes Selected:[DataNode1, DataNode3, DataNode2]
Block:1  Data Nodes Selected:[DataNode4, DataNode1, DataNode3]
Block:2  Data Nodes Selected:[DataNode2, DataNode4, DataNode1]
Block:3  Data Nodes Selected:[DataNode3, DataNode2, DataNode4]
Block:4  Data Nodes Selected:[DataNode1, DataNode3, DataNode2]
Block:5  Data Nodes Selected:[DataNode4, DataNode1, DataNode3]
Block:6  Data Nodes Selected:[DataNode2, DataNode4, DataNode1]
Block:7  Data Nodes Selected:[DataNode3, DataNode2, DataNode4]
Block:8  Data Nodes Selected:[DataNode1, DataNode3, DataNode2]
Block:9  Data Nodes Selected:[DataNode4, DataNode1, DataNode3]
Block:10  Data Nodes Selected:[DataNode2, DataNode4, DataNode1]
Block:11  Data Nodes Selected:[DataNode3, DataNode2, DataNode4]
Block:12  Data Nodes Selected:[DataNode1, DataNode3, DataNode2]
Block:13  Data Nodes Selected:[DataNode4, DataNode1, DataNode3]
Block:14  Data Nodes Selected:[DataNode2, DataNode4, DataNode1]
Block:15  Data Nodes Selected:[DataNode3, DataNode2, DataNode4]
Block:16  Data Nodes Selected:[DataNode1, DataNode3, DataNode2]
Block:17  Data Nodes Selected:[DataNode4, DataNode1, DataNode3]
Block:18  Data Nodes Selected:[DataNode2, DataNode4, DataNode1]
Block:19  Data Nodes Selected:[DataNode3, DataNode2, DataNode4]
Block:20  Data Nodes Selected:[DataNode1, DataNode3, DataNode2]
```

Figure10: *Installing four DataNodes in HDFS*

While reading a files in HDFS, the clientcreates a local file and DFS File. To upload a file to DFS a DataNode is created and registered with the rmiregistry. It can Access the NameNode and register with the NameNode to indicate it is available and returns the number of bytes read.

Using the selection policy as 'first', 'random' and 'loadbased' perform 20, 25, 30, 35, 40, 45 and 50 downloads concurrently. The number of concurrent downloads initiated: 20, 25, 30, 35, 40, 45, and 50. Policies experimented to select the server: first, random and loadbased. Select the first, random and loadbased server during the block retrieval for 20 concurrent requests. In the same way select the server during the block retrieval for 25, 30, 35, 40, 45 and 50 concurrent requests. Use block size as complete file size, with fault-tolerance level 2 (the complete file replicated at 3 servers). Create DFS File and set block size as 1K, 2K, 5K, 10K, 25K, 50K, 100K and the following. Set block size as 1K in constants file to test the download time for 1K block size. In the same way test the download time for 2K, 5K, 10K, 25K, 50K and 100K. The below tables contains the values of average requests execution time for first, random and loadbased.

Load entries from DataNode Metadata location to files on system. Save entries from files on system to DataNode Metadata location. Find size of the file. Use block size to find how many blocks needed. Use DataNodes in the DataNode map. Copy each block to the number of DataNodes (based on fault level). Block number starts from 0 to blockCt-1. In DataNodeMetadata the blocks are chopping into DataNodes as shown in below format.



PC › Windows (C:) › dfs › DataNodeMetadata › DataNode1

Name	Date modified	Type	Size
DATANODE_METADATA	15-03-2017 12:50	File	166 KB
DFSfile_BLOCK_0	15-03-2017 12:21	File	5,000 KB
DFSfile_BLOCK_1	15-03-2017 12:21	File	5,000 KB
DFSfile_BLOCK_2	15-03-2017 12:21	File	5,000 KB
DFSfile_BLOCK_4	15-03-2017 12:21	File	5,000 KB
DFSfile_BLOCK_5	15-03-2017 12:21	File	5,000 KB
DFSfile_BLOCK_6	15-03-2017 12:21	File	5,000 KB
DFSfile_BLOCK_8	15-03-2017 12:21	File	5,000 KB
DFSfile_BLOCK_9	15-03-2017 12:22	File	5,000 KB
DFSfile_BLOCK_10	15-03-2017 12:22	File	5,000 KB
DFSfile_BLOCK_12	15-03-2017 12:22	File	5,000 KB
DFSfile_BLOCK_13	15-03-2017 12:22	File	5,000 KB
DFSfile_BLOCK_14	15-03-2017 12:22	File	5,000 KB
DFSfile_BLOCK_16	15-03-2017 12:22	File	5,000 KB
DFSfile_BLOCK_17	15-03-2017 12:22	File	5,000 KB
DFSfile_BLOCK_18	15-03-2017 12:22	File	5,000 KB
DFSfile_BLOCK_20	15-03-2017 12:22	File	2,400 KB

Figure11: *shows blocks distributed over the DataNodes in HDFS*

	first	random	loadbased
20	63950	72393	29101
25	86762	83665	31454
30	82932	33617	43523
35	79521	77924	43696
40	78193	76585	47890
45	77157	75192	48670
50	90023	65924	46455

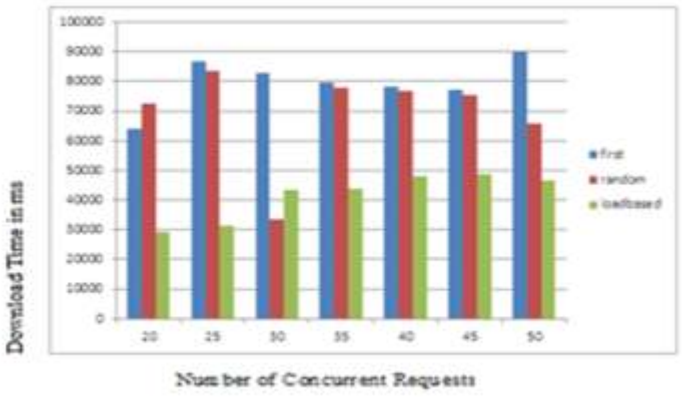
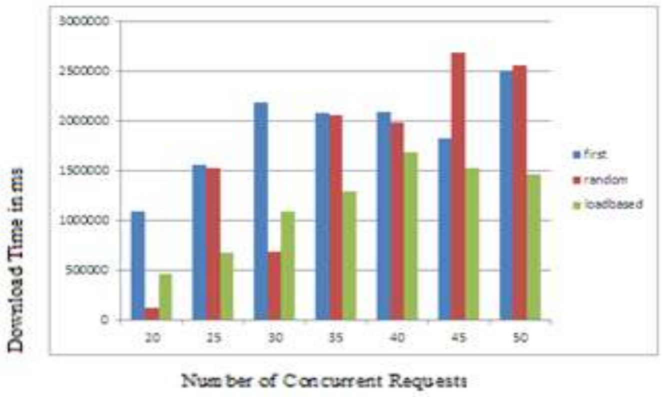


Figure12: *Download time taken to read a file*

	first	random	loadbased
20	1084468	121185	460396
25	1553222	1528609	678011
30	2181309	680139	1092919
35	2081565	2053479	1287607
40	2090863	1978283	1684813
45	1823770	2680788	1529965
50	2501479	2559501	1456254



	first	random	loadbased
20	54223.4	60909.25	23019.8
25	62128.88	61144.36	27120.44
30	72710.3	22671.3	36430.63
35	59473.29	58670.83	36788.77
40	52271.57	49457.07	42120.32
45	40528.22	59573.07	33999.22
50	50029.58	51190.02	33999.22

Figure13: *Total Execution Time for required requests*

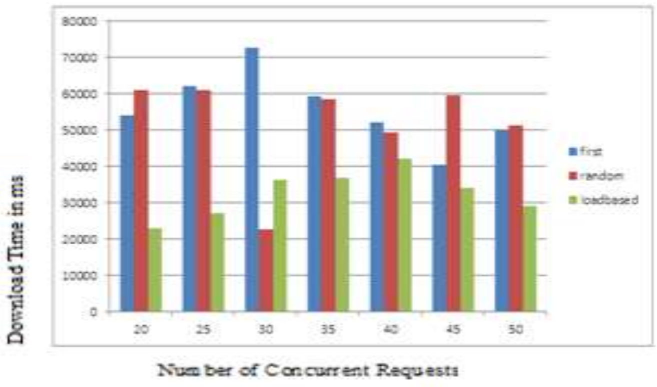


Figure14: *Average requests Execution Time*

Figures describe three policies i.e., first, random and loadbased. We observe the figure12, download time to read a file inHDFS, the policy is faster than first and random and if weobserve figure13 and figure14 the total execution time for therequired requests and average requests execution time in HDFS, the first and random policies areslower than loadbased. If we compare the three policies we can observe that the speed of loadbased is better and faster than remaining two cases.

VIII. CONCLUSION

Hadoop distributed file system provides a high throughput access to the data of an application and is suitable for applications that needs to work with large datasets. HDFS designed to carry petabytes of data with high fault tolerance. Petabytes of data are saved redundantly over many servers or machines. Files containing data are stored redundantly across number of machines for high availability and durability to failure. The block size setting is used by HDFS to divide files into blocks and then distribute those blocks across the cluster. We have examined the design and architecture of Hadoop distributed file system. Particularly our analysis focus on three policies those are first, random and loadbased. Observes the results comparison between three policies the speed of download time to read a file in HDFS is faster in loadbased. HDFS is used for storage and provides more efficiency to the system.

REFERENCES

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

[2] Tom White, “Hadoop The Definitive Guide”, 2nd ed., O’REILLY, 2011, pp. 41–73.

[3] https://en.wikipedia.org/wiki/Apache_Hadoop

[4] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler,“Hadoop Distributed File System”, 2010

[5] Towards Better Fault Tolerance in HDFS – A Survey Paper, IJIRT 2014.

[6] https://technocents.wordpress.com/category/hadoop/hadoop- distributed-file-system-hdfs/

[7] The Hadoop Distributed File System: Architecture and Design” by Dhruba Borthakur, http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf

[8] Keerthivasan M., “Review of Distributed File Systems: Concepts and Case Studies”, ECE 677 Distributed Computing Systems

[9] Pooja S.Honnutagi, “The Hadoop distributed file system”, International Journal of Computer Science and Information Technology, Vol.5(5), 2014, pp. 6238-6243

[10] Jaskaran Singh, Varun Singla “Big Data: Tools and Technologies in Big Data”. In International Journal of Computer Applications (0975 – 8887) Volume 112 – No 15, February 2015.

[11] Puneet Singh Duggal, Sanchita Paul ,―BigDataAnalysis: Challenges and Solutions‖, International Conference on Cloud, Big Data and Trust 2013, Nov 13-15, RGPV

[12] Kiran kumara Reddi & Dnvsl Indira “Different Technique to Transfer Big Data : survey” IEEE Transactions on 52(8) (Aug.2013) 2348 { 2355}

[13] http://www.informit.com/articles/article.aspx?p=2460260& seqN=2

[14] S.Vikram Phaneendra & E.Madhusudhan Reddy “Big Data- solutions for RDBMS problems- A survey” In 12th IEEE/IFIP Network Operations & Management Symposium (NOMS 2010) (Osaka, Japan, Apr 19{23 2013).

[15] Bappalige, S.P. An introduction to Apache Hadoop for big data. Retrieved Nov 27, 2014, available at http://opensource.com/life/14/8/introapache-hadoop- bigdata

[16] Jonathan Stuart Ward and Adam Barker “Undefined By Data: A Survey of Big Data Definitions”Stamford, CT: Gartner, 2012.