An Effective Big Data Role of Data Storage and Job Tracking Analysis through the HDFS

S.Janardhan[1], Rajeshkumar.P[2], B.Madhu Sudhan Reddy[3]

[1]*Student, Master of Computer Applications, SKIIMS, Srikalahasti, Andhra Pradesh India*
[2]*Research Scholar, Computer science, Bharathiar University, Coimbatore, India*
[3]*Asst.Professor, Master of Computer Applications, SKIIMS, Srikalahasti, India*

***Abstraction*: - Bigdata inspires new ways to transform processes, organizations, entire industries and even society itself. Yet extensive media coverage makes it hard to distinguish hype from reality. Big data is a collection of massive and complex data sets that include the huge quantities of data, social media analytics, data management capabilities, real-time data. Big Data includes e-mail messages, snaps, business deals, surveillance video recordings, posts of social Medias, mobile phone GPS signals and RFID readers, microphones, cameras, sensors and activity logs. That's why, the data that exceeds the processing capacity of conventional database systems. Big data processing such as Hadoop which uses the map-reduce paradigm. Using MapReduce programming paradigm, the big data is processed. Whatever the label, organizations are starting to understand and explore how to process and analyze a vast array of information in new ways. This paper will explain about HDFS architecture, high volume of data storage and query processing.**

***Key Words:* Hadoop, HDFS, Name Node, Secondary Name Node, JobTracker, DataNode, Task Tracker**

I. INTRODUCTION

Hadoop is an ecosystem of open source components that fundamentally changes the way enterprises store, process, and analyzes data. Unlike traditional systems, Hadoop enables multiple types of analytic workloads to run on the same data, at the same time, at massive scale on industry-standard hardware. CDH, Cloudera's open source platform, is the most popular distribution of Hadoop and related projects in the world (with support available via a Cloudera Enterprise subscription). The Hadoop Distributed File System (HDFS) is a distributed file system 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. HDFS[1] is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nautch web search engine project. HDFS is now an Apache Hadoop subproject. Hadoop provides a distributed filesystem and a framework for the analysis and transformation of very large data sets using the MapReduce paradigm. While the interface to HDFS is patterned after the UNIX filesystem, faithfulness to standards was sacrificed in favor of improved performance for the applications at hand. An important characteristic of Hadoop is the partitioning of data and computation across many (thousands) of hosts, and the execution of application computations in parallel close to their data. A Hadoop cluster scales computation capacity, storage capacity and I/O bandwidth by simply adding commodity servers. Hadoop clusters at Yahoo! span 40,000 servers, and store 40 petabytes of application data, with the largest cluster being 4000 servers. One hundred other organizations worldwide report using Hadoop. HDFS stores filesystem metadata and application data separately. As in other distributed filesystems, like PVFS, Lustre2, and GFS, HDFS stores metadata on a dedicated server, called the Name Node. Application data are stored on other servers called Data Nodes. All servers are fully connected and communicate with each other using TCP-based protocols. Unlike Luster and PVFS, the Data Nodes in HDFS do not rely on data protection. Mechanisms such as RAID[3] to make the data durable. Instead, like GFS, the file content is replicated on multiple Data Nodes for reliability. While ensuring data durability, this strategy has the added advantage that data transfer bandwidth is multiplied, and there are more opportunities for locating computation near the needed data.

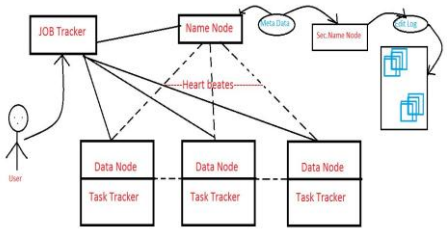


Fig1: HDFS Architecture

II. HDFS ARCHITECTURE

Hadoop Distributed File System or Master Slave Architecture. +In HDFS we have 5 Components or Demon's

2.1 Name Node

The HDFS namespace is a hierarchy of files and directories. Files and directories are represented on the Name Node by Inodes. Inodes record attributes like permissions, modification and access times, namespace and disk space quotas. The file content is split into large blocks (typically 128 megabytes, but

user selectable file-by-file), and each block of the file is independently replicated at multiple Data Nodes (typically three, but user selectable file-by-file). The Name Node maintains the namespace tree and the mapping of blocks to Data Nodes. The current design has a single Name Node for each cluster. The cluster can have thousands of Data Nodes and tens of thousands of HDFS clients per cluster, as each Data Node may execute multiple application tasks concurrently. In Name Node the status of Data Node and Task Tracker is maintained in Meta Data internally all Nodes are connected these connection is called Data Localization.

2.2 Secondary Name Node

The Name Node stores modifications to the file system as a log appended to a native file system file, edits. When a Name Node starts up, it reads HDFS state from an image file, fsimage, and then applies edits from the edits log file. It then writes new HDFS state to the fsimage and starts normal operation with an empty edits file. Since Name Node merges fsimage and edits files only during start up, the edits log file could get very large over time on a busy cluster. Another side effect of a larger edits file is that next restart of Name Node takes longer. The secondary Name Node merges the fsimage and the edits log files periodically and keeps edits log size within a limit. It is usually run on a different machine than the primary Name Node since its memory requirements are on the same order as the primary NameNode. The start of the checkpoint process on the secondary Name Node is controlled by two configuration parameters.

- dfs. namenode. check point .period, set to 1 hour by default, specifies the maximum delay between two consecutive checkpoints, and
- dfs.namenode.checkpoint.txns, set to 1 million by default, defines the number of uncheck pointed transactions on the Name Node which will force an urgent checkpoint, even if the checkpoint period has not been reached.

The secondary Name Node stores the latest checkpoint in a directory which is structured the same way as the primary Name Node's directory. So that the check pointed image is always ready to be read by the primary Name Node if necessary. in Secondary Name Node provide the back up of Name Node and it can store the text data into fsimages.the fsimages can convert into the text format data to the fsimages format it is called EDIT LOG. It's software to convert the text data to image format and image format to text format.

2.3 Job Tracker

JobTracker and Task Tracker are[4] 2 essential process involved in MapReduce execution in MRv1 (or Hadoop version 1). Both processes are now deprecated in MRv2 (or Hadoop version 2) and replaced by Resource Manager, Application Master and Node Manager Daemons.

1. Job Tracker process runs on a separate node and not usually on a Data Node.
2. Job Tracker is an essential Daemon for MapReduce execution in MRv1. It is replaced by Resource Manager/Application Master in MRv2.
3. Job Tracker receives the requests for MapReduce execution from the client.
4. Job Tracker talks to the Name Node to determine the location of the data.
5. Job Tracker finds the best Task Tracker nodes to execute tasks based on the data locality (proximity of the data) and the available slots to execute a task on a given node.
6. Job Tracker monitors the individual Task Trackers and the submits back the overall status of the job back to the client.
7. Job Tracker process is critical to the Hadoop cluster in terms of MapReduce execution.
8. When the Job Tracker is down, HDFS will still be functional but the MapReduce execution cannot be started and the existing MapReduce jobs will be halted.

2.4 Data Node

Each block replica on a Data Node is represented by two files in the local native filesystem. The first file contains the data itself and the second file records the block's metadata including checksums for the data and the generation stamp. The size of the data file equals the actual length of the block and does not require extra space to round it up to the nominal block size as in traditional filesystems. Thus, if a block is half full it needs only half of the space of the full block on the local drive. During startup each Data Node connects to the Name Node and performs a handshake. The purpose of the handshake is to verify the namespace ID[7] and the software version of the Data Node. If either does not match that of the Name Node, the Data Node automatically shuts down. The namespace ID is assigned to the filesystem instance when it is formatted. The namespace ID is persistently stored on all nodes of the cluster. Nodes with a different namespace ID will not be able to join the cluster, thus protecting the integrity of the filesystem. A Data Node that is newly initialized and without any namespace ID is permitted to join the cluster and receive the cluster's namespace ID. After the handshake the Data Node registers with the Name Node. Data Nodes persistently store their unique storage IDs. The storage ID is an internal identifier of the Data Node, which makes it recognizable even if it is restarted with a different IP address or port. The storage ID is assigned to the Data Node when it registers with the Name Node for the first time and never changes after that. A Data Node identifies block replicas in its possession to the Name Node by sending a block report. A block report contains the block ID, the generation stamp and the length for each block replica the server hosts. The first block report is sent immediately after the Data Node registration. Subsequent block reports are sent every hour and provide the Name Node with an up-to-date view of where

block replicas are located on the cluster. During normal operation Data Nodes send heartbeats to the Name Node to confirm that the Data Node is operating and the block replicas it hosts are available. The default heartbeat interval is three seconds. If the Name Node does not receive a heartbeat from a Data Node in ten minutes the Name Node considers the Data Node to be out of service and the block replicas hosted by that Data Node to be unavailable. The Name Node then schedules creation of new replicas of those blocks on other Data Nodes. Heartbeats from a Data Node also carry information about total storage capacity, fraction of storage in use, and the number of data transfers currently in progress. These statistics are used for the Name Node's block allocation and load balancing decisions. The Name Node does not directly send requests to Data Nodes. It uses replies to heartbeats to send instructions to the Data Nodes. The instructions include commands to replicate blocks to other nodes, remove local block replicas, re-register and send an immediate block report, and shut down the node. These commands are important for maintaining the overall system integrity and therefore it is critical to keep heartbeats frequent even on big clusters. The Name Node can process thousands of heartbeats per second without affecting other Name Node operations.

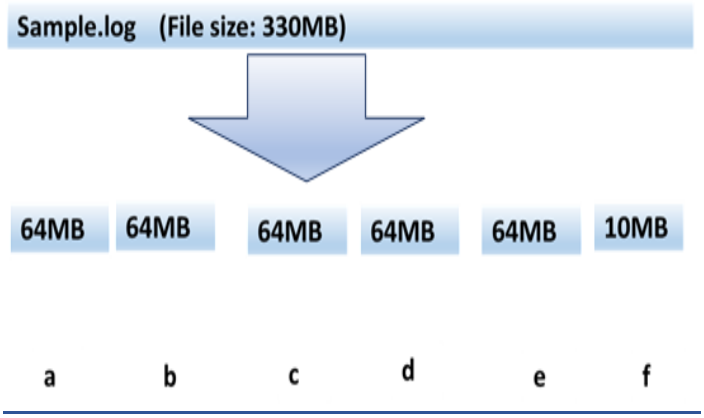
2.5 Task Tracker

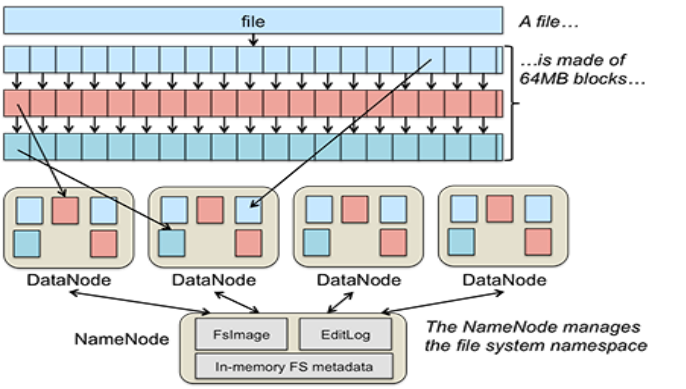
A Task Tracker is a node in the cluster that accepts tasks - Map, Reduce and Shuffle operations - from a Job Tracker. Every Task Tracker is configured with a set of slots; these indicate the number of tasks that it can accept. When the Job Tracker tries to find somewhere to schedule a task within the MapReduce operations, it first looks for an empty slot on the same server that hosts the Data Node containing the data, and if not, it looks for an empty slot on a machine in the same rack. The Task Tracker spawns a separate JVM [10] processes to do the actual work; this is to ensure that process failure does not take down the task tracker. The Task Tracker monitors these spawned processes, capturing the output and exit codes. When the process finishes, successfully or not, the tracker notifies the Job Tracker. The Task Trackers also send out heartbeat messages to the Job Tracker, usually every few minutes, to reassure the Job Tracker that it is still alive. These messages also inform the Job Tracker of the number of available slots, so the Job Tracker can stay up to date with where in the cluster work can be delegated. in this Job Tracker performing the particular task after completion of task it will send the status of task tracker then Name node assign to the next work these process will be continue until finish the work. Every three seconds Data Node and Task Tracker send the report or update when am free or not what work am doing now to send a report to the Name Node.

III. DATA STORAGE IN HDFS

- Data storage in HDFS is different from the way files are stored in Windows or Linux.
- In HDFS, the system breaks down the file meant for storage into a set of Individual blocks and later stores the blocks in various slave nodes of the Hadoop cluster.
- Minimum size of block is 64MB and Maximum size can be integral (excluding zero and negative numbers) multiple of 64MB

Example: A file (Sample.log) being divided into blocks of data (a,b,c,d,e,f)



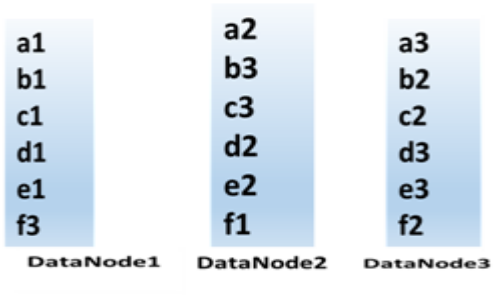


3.1 Replicating data blocks

- HDFS was designed to store the data even in inexpensive commodity hardware. The hardware can be sometimes unreliable. So to overcome any hardware failures, HDFS replicates data blocks that are stored in the system.
- Note: The norm is three copies of individual data blocks and any order can be stored in the data node. However, no two blocks of same type are stored in the data node (i.e. a1, a2 in the same data node 1).

From the above example the file (Sample.log) being divided into blocks of data (a,b,c,d,e,f). Which in turn are

replicated in to three (a1,b1,c1,d1,e1,f1). (a2,b2,c2,d2,e2,f2) and (a3,b3,c3,d3,e3,f3)



3.2 Name Node Memory concerns

In order to scale the name service horizontally, federation uses multiple independent Namenodes/namespaces. The Namenodes are federated; the Namenodes are independent and do not require coordination with each other. The Datanodes are used as common storage for blocks by all the Namenodes. Each Datanode registers with all the Namenodes in the cluster. Datanodes send periodic heartbeats and block reports. They also handle commands from the Namenodes.

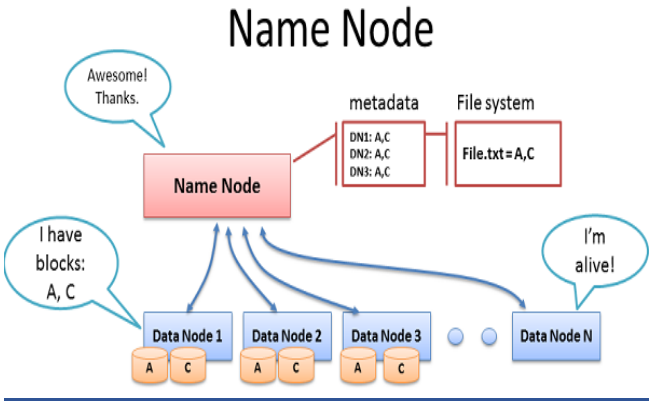
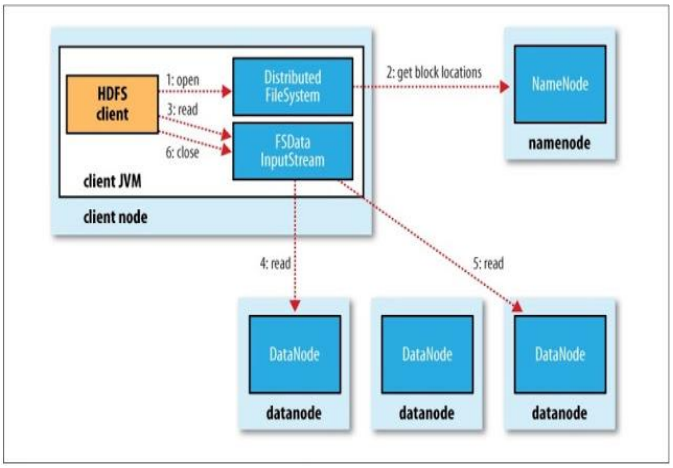


Fig2: name nodes

3.3 Anatomy of File Read

In the following section we are going to understand the flow of HDFS file read operation. Let's have a look at pictorial representation of the data read operation and components involved in it.



HDFS Client: HDFS client interacts with Namenode and Datanode to fulfil user request. User establishes communication with HDFS through File System API and normal I/O operations, processing of user request and providing response over it is carried out by File System API processes.

NameNode: It is the MasterNode. It stores metadata information contains addresses of block locations of Datanodes, this information is used for file read and write operation to access the blocks in a HDFS cluster.

Datanode: Datanode also known as slave node holds the actual data. Datanode only stores block, a block is what is used to store and process the data.Datanode gives periodic heartbeat signals to Masternode to indicate that it is alive and can be used to store and retrieve data

Packet: A Packet is a small chunk of data which is used during transmission; packet is a subset of block. The default size of a block is around 64 MB or 128 MB, it will create a huge network overload if we transfer data of the size of blocks, hence client API transfers this block in small chunks known as packets.

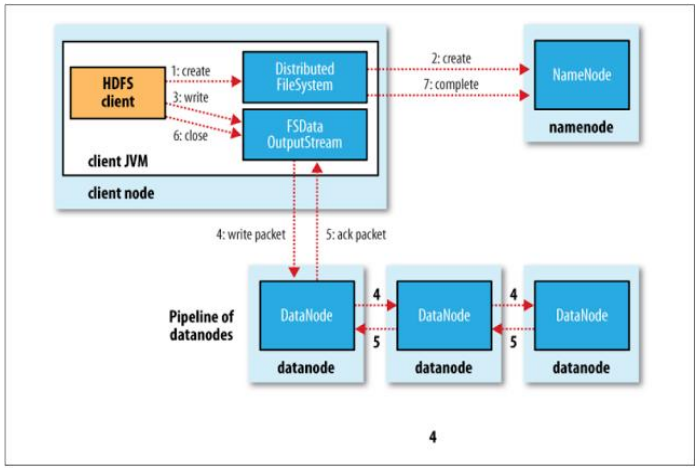
3.4 Anatomy of File Write

Now we will look at what happens when you write a File in HDFS.

Before client start writing data to HDFS it grabs an instance of an object of Distributed File System (HDFS)

Distributed File System object do a RPC call to namenode to create a new file in filesystem namespace with no blocks associated to it Namenode process performs various checks like

a) client has required permissions to create a file or not
b) file should not exists earlier. In case it will throw an IOexception to client



Once the file is registered with the namenode then client will get an object i.e. FSDataOutputStream

Which in turns embed DFSOutputStream object for the client to start writing data to DFSoutputStream handles communication with the datanodes and namenode.

As client writes data DFSOutputStream split it into packets and write it to its internal queue i.e. data queue and also maintains an acknowledgement queue.

Data queue is then consumed by a Data Streamer process which is responsible for asking namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas.

The list of datanodes forms a pipeline and assuming a replication factor of three, so there will be three nodes in the pipeline.

The data streamer streams the packets to the first datanode in the pipeline, which then stores the packet and forwards it to second datanode in the pipeline. Similarly the second node stores the packet and forwards it to next datanode or last datanode in the pipeline

Once each datanode in the pipeline acknowledge the packet the packet is removed from the acknowledgement queue.

IV. HDFS COMMANDS

The File System (FS) shell includes various shell-like commands that directly interact with the Hadoop Distributed File System (HDFS) as well as other file systems that Hadoop supports.

The fs shell can be invoked by $ bin/hdfs dfs <args>

Most of the commands in FS shell behave like corresponding Unix commands.

$cat Syntax: hdfs dfs -cat <<File path>>

Usage : Lists contents of the file.

$chgrp Syntax: hdfs dfs –chgrp <<group name>><<file path>>

Usage : Lists contents of the file.

$**chmod** Syntax: hdfs dfs –chmod <<octal number>><<file path>>

Usage: 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.

$**chown** Syntax: hdfs dfs –chown <<Owner name>><<file path>>

Usage: Change the owner of files. With -R, make the change recursively through the directory structure. The user must be a super-user.

$**copyFromLocal** Syntax: hdfs dfs –copyFromLocal

<<local input directory path>><< output hdfs directory path >>

Note: one can use put instead of copyFromLocal

Usage: Copies file from the local source to a output hdfs directory.

$ **copyToLocal** Syntax: hdfs dfs –copyToLocal

<< input hdfs directory path >><<local input directory path>>

Note: one can use get instead of copyToLocal

Usage: Copies file from the local hdfs directory to a output local directory.

$**count** Syntax: hdfs dfs –count <<directory path>>

Usage: Count the number of directories, files and bytes under the paths thatmatch the specified file pattern.

cp Syntax: hdfs dfs –cp <<source hdfs path>><<output hdfs path>>

Usage: Copy files from source hdfs path to destination hdfs path.

$du Syntax: hdfs dfs -du <<directory path>>

Usage: Displays sizes of files and directories contained in the given directory orthe length of a file in case it's just a file.

$**mkdir** Syntax:hdfs dfs -mkdir <<directory name>>

Usage: Creates a directory

$ **ls** Syntax: hdfs dfs -ls <<directory or file>>

Usage:For a file or directory returns the statistics.

moveFromLocal Syntax: dfs -moveFromLocal <<local file path>><<output hdfspath>>

Similar to **put**command, except that the source file is deleted after it's copied.

moveToLocal Syntax:hdfs dfs -moveToLocal <<input hdfs path>><<output localfile path>>

isplays a "Not implemented yet" message.

$mv Syntax: hdfs dfs -mv <<input file path>><<output file path>>

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 file systems is not permitted.

$**rm** Syntax: hdfs dfs -rm <<File path>>

Delete the specified file

$**setrep** Syntax: hdfs dfs -setrep [-R] <<File path>>

Usage: Changes the replication factor of a file. -R option is for recursively increasing the replication factor of files within a directory.

$**fsck** Syntax: hdfs fsck –blocks <<file path>>

Usage: fsck – file system check

V. HDFS ADMINISTRATIVE COMMANDS

Hadoop in safe mode means:

- Name Node will be in safe mode and HDFS will be in read only mode There are four important HDFS administrative commands

1) $hdfs dfsadmin –safemode get

Usage: Gives the status of hdfs system. whether Hadoop is operating in safemode or not (i.e. either on or off state).

2) $hdfs dfsadmin –safemode enter

Usage: Hadoop operates in safemode (i.e. on state)

3) $hdfs dfsadmin –safemode leave

Usage: Hadoop leaves safemode (i.e. off state)

4) $hdfs dfsadmin –report

Usage: Gives report of the Hadoop system

VI. CONCLUSION

In This Article an overview of HDFS architecture Name Node, Secondary Name Node, Job Tracker, Data Node, Task Tracker have been reviewed. The results showing quickly and processing the Data very fast it will take query and gives the particular related information with in short time. in HDFS Every query we are giving that is small or big it will take same time and gives output with in time and the storage is very large and scalable, processing the data very fast in the mean of time frequency is low to compare other one's HDFS is taking time to Process the query very fast in HDFS the data will be sharing or Distributed and processing the data this is major point in HDFS and cost also low in future days the data storage will expandable very high and improve ,develop the query processing.

REFERENCES

[1]. Sagiroglu, Sinanc, "Big Data: A Review", 978-1-4673-6404-1/13 IEEE.
[2]. Sabia, Arora, "Technologies to Handle Big Data: A Survey".
[3]. Shilpa, Manjit Kaur, "Big Data and Methodology – A review", Volume 3, Issue 10, October 2013.
[4]. http://www-01.ibm.com/software/in/data/bigdata/ (last access - 10/11/2014).
[5]. K. Bakshi, "Considerations for Big Data: Architecture and Approach", Aerospace Conference IEEE, Big Sky Montana, March 2012.
[6]. D. Garlasu, V. Sandulescu, I. Halcu, G.Neculoiu, "A Big Data implementation based on Grid Computing", Grid Computing, 17-19 January 2013.
[7]. R.D. Schneider, "Hadoop for Dummies Special Edition", John Wiley&Sons Canada, 978-1-118-5051-8, 2012.
[8]. Yuri Demchenko, "The Big Data Architecture Framework (BDAF)", Outcome of the Brainstorming Session at the University of Amsterdam, 17 July 2013.
[9]. Aditya B. Patel, Manashvi Birla, Ushma Nair, "Addressing Big Data Problem Using Hadoop and Map Reduce", Aug, 2012.
[10]. http://removeandreplace.com/2013/03/13/how-much-data-is-on-the-internet-and-generated-online-every-minute/*Query:* how much data is on the internet and generated online every minute.
[11]. http://bigdatauniversity.com/
[12]. http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html (last access 4/11/2014).
[13]. http://www.mssqltips.com/sqlservertip/3222/big-data-basics--part-5--introduction-to-mapreduce/ (last access 29/10/2014).