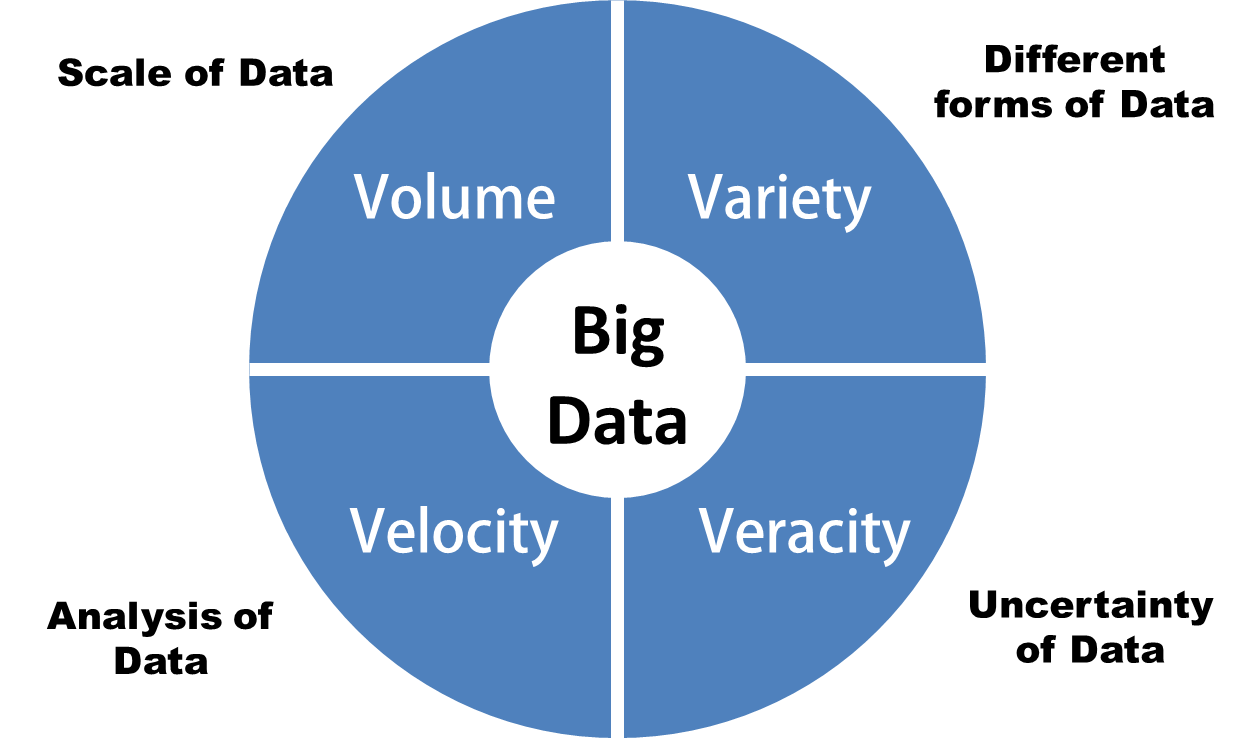
**CHAPTER 1**

**INTRODUCTION OF BIG DATA**

Big data is a term for [data sets](https://en.wikipedia.org/wiki/Data_set) that are so large or complex that traditional [data processing](https://en.wikipedia.org/wiki/Data_processing) applications are inadequate. Data sets are growing rapidly in part because they are increasingly gathered by cheap and numerous information-sensing [mobile devices](https://en.wikipedia.org/wiki/Mobile_device), remote sensing, software logs, [cameras](https://en.wikipedia.org/wiki/Digital_camera), microphones, [radio-frequency identification](https://en.wikipedia.org/wiki/Radio-frequency_identification) (RFID) readers and [wireless sensor networks](https://en.wikipedia.org/wiki/Wireless_sensor_networks).

The world's technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s; as of 2016, every day 2.5 [Exabyte’s](https://en.wikipedia.org/wiki/Exabyte) (2.5×1018) of data is created. One question for large enterprises is determining who should own big data initiatives that affect the entire organization.

Challenges caused by big data include capture, curation, storage, search, sharing, transfer, analysis and visualization.

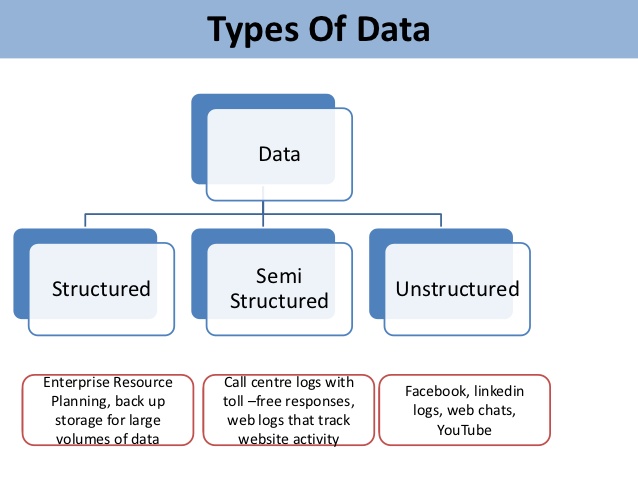


**Figure 1.1 Characteristic of Big Data**

* Volume: big data doesn't sample; it just observes and tracks what happens
* Variety: big data draws from text, images, audio, video; plus it completes missing pieces through data fusion.
* Veracity: Uncertainty of Data.
* Velocity: big data is often available in real-time
  1. **Types of Data:**

There are basically three types of data

* Structured
* Semi structured
* Unstructured

****

**Figure 1.2 Types of Data**

**Structured Data:**

 It concerns all data which can be stored in database SQL in table with rows and columns. They have relational key and can be easily mapped into pre-designed fields. Today, those data are the most processed in development and the simplest way to manage information.

**Semi Structured Data:**

 Semi-structured data is information that doesn’t reside in a relational database but that does have some organizational properties that make it easier to analyse.

Examples of semi structured: CSV but XML and JSON documents are semi structured documents, NoSQL databases are considered as semi structured.

**Unstructured Data:**

Unstructured data represent around 80% of data. It often include text and multimedia content. Examples include e-mail messages, word processing documents, videos, photos, audio files, presentations, webpages and many other kinds of business documents.

**CHAPTER 2**

**INTRODUCTION OF HADOOP**

Today, we’re surrounded by data. People upload videos, take pictures on their cell phones, text friends, update their Facebook status, leave comments around the web, click on ads, and so forth. Machines, too, are generating and keeping more and more data. The exponential growth of data first presented challenges to cutting-edge businesses such as Google, Yahoo, Amazon, and Microsoft. They needed to go through terabytes and petabytes of data to figure out which websites were popular, what books were in demand, and what kinds of ads appealed to people. Existing tools were becoming inadequate to process such large data sets. Google was the first to publicize Map Reduce—a system they had used to scale their data processing needs. This system aroused a lot of interest because many other businesses were facing similar scaling challenges, and it wasn’t feasible for everyone to reinvent their own proprietary tool. Doug Cutting saw an opportunity and led the charge to develop an open source version of this Map Reduce system called Hadoop. Soon after, Yahoo and others rallied around to support this effort. Today, Hadoop is a core part of the computing infrastructure for many web companies, such as Yahoo, Facebook, LinkedIn, and Twitter. Many more traditional businesses, such as media and telecom, are beginning to adopt this system too. Hadoop is an open source framework for writing and running distributed applications that process large amounts of data.

Distributed computing is a wide and varied field, but the key distinctions of Hadoop are that it is

■ **Robust**—because it is intended to run on commodity hardware, Hadoop is architected with the assumption of frequent hardware malfunctions. It can gracefully handle most such failures.

■ **Scalable**—Hadoop scales linearly to handle larger data by adding more nodes to the cluster.

■ **Simple**—Hadoop allows users to quickly write efficient parallel code. Hadoop’s accessibility and simplicity give it an edge over writing and running large distributed programs. Even college students can quickly and cheaply create their own Hadoop cluster. On the other hand, its robustness and scalability make it suitable for even the most demanding jobs at Yahoo and Facebook. These features make Hadoop popular in both academia and industry.

**2.1 HISTORY OF HADOOP**

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project. The Origin of the Name “Hadoop”: The name Hadoop is not an acronym; it’s a made-up name. The project’s creator, Doug Cutting, explains how the name came about:

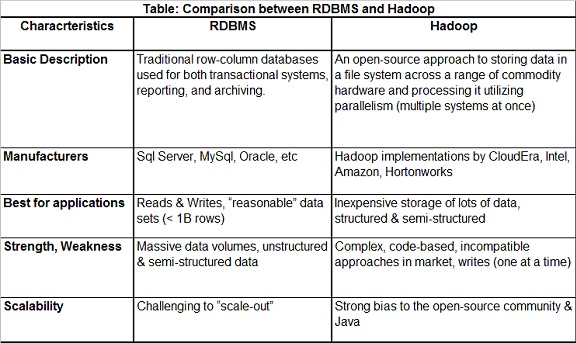
The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid’s term.

Subprojects and “contribute” modules in Hadoop also tend to have names that are unrelated to their function, often with an elephant or other animal theme (“Pig” for example). Smaller components are given more descriptive (and therefore more mundane) names. This is a good principle, as it means you can generally work out what something does from its name. For example, the job tracker keeps track of Map Reduce jobs.

Building a web search engine from scratch was an ambitious goal, for not only is the software required to crawl and index websites complex to write, but it is also a challenge torun without a dedicated operations team, since there are so many moving parts. It’s expensive too: Mike Cafarella and Doug Cutting estimated a system supporting a 1- billion-page index would cost around half a million dollars in hardware, with a monthly running cost of $30,000.Nevertheless, they believed it was a worthy goal, as it would open up and ultimately democratize search engine algorithms. Nutch was started in 2002, and a working crawler and search system quickly emerged. However, they realized that their architecture wouldn’t scale to the billions of pages on the Web. Help was at hand with the publication of a paper in 2003 that described the architecture of Google’s distributed file system, called GFS, which was being used in production at Google.# GFS, or something like it, would solve their storage needs for the very large files generated as a part of the web crawl and indexing process. In particular, GFS would free up time being spent on administrative tasks such as managing storage nodes. In 2004, they set about writing an open source implementation, the Nutch Distributed File system (NDFS). In 2004, Google published the paper that introduced Map Reduce to the world. Early in 2005, the Nutch developers had a working Map Reduce implementation in Nutch, and by the middle of that year all the major Nutch algorithms had been ported to run using Map Reduce and NDFS. NDFS and the Map Reduce implementation in Nutch were applicable beyond the realm of search, and in February 2006 they moved out of Nutch to form an independent subproject of Lucene called Hadoop. At around the same time, Doug Cutting joined Yahoo!, which provided a dedicated team and the resources to turn Hadoop into a system that ran at web scale (see sidebar). This was demonstrated in February 2008 when Yahoo! Announced that its production search index was being generated by a 10,000-core Hadoop cluster. In January 2008, Hadoop was made its own top-level project at Apache, confirming its success and its diverse, active community. By this time Hadoop was being used by many other companies besides Yahoo!, such as Last.fm, Facebook, and the New York Times.

**2.2 DIFFERENCE BETWEEN RDBMS AND HADOOP:**

Traditional RDBMS is used for transactional systems to report and archive the data, whereas Hadoop is an approach to store huge amount of data in the distributed file system and process it. RDBMS works on structured data unlike Hadoop works on unstructured data. RDBMS stores the data in the form of rows and columns where as it stores the data in the form of DFS.  RDBMS will be useful when you want to seek one record from big data, whereas, Hadoop will be useful when you want Big data in one shot and perform analysis.



**Fig 2.1 Comparison between RDBMS and Hadoop**

# 2.3 PREREQUISITES:

Before installing Hadoop and running Hadoop services, make sure that your system meets following requirements:

* + Linux OS
  + Java installed on the system.
  + SSH configured on the system.
    - ssh-keygen
    - give no passphrase
    - ssh-copy-id –i localhost
* Follow the steps below to install 2.x on a **single node** linux machine:

1. Create a folder for Hadoop in “opt” say /opt/setups/
2. If you do not have permission for creating folder in opt, you could execute following commands to change the mode of “opt” folder:
   1. sudo chmod -R xxx /opt/ (here xxx could be the linux persmission set like 777).
3. cd /opt/setups.
4. Download 2.x version tar.gz file of Hadoop from [here](http://www.bizdirusa.com/mirrors/apache/hadoop/common/).
5. .Untar the file to a folder /opt/ using command:
   1. tar -xzvf hadoop-filename.tar.gz
6. Move hadoop setup folder to /opt/
   1. mv hadoop-2.6.0 /opt/
7. cd /opt/hadoop-2.6.0
8. Before installing Hadoop, configure the environment variables: open or create bash profile in folder of user: $ nano ~/.bash\_profile

Now enter following env. Variables in this file:

         export HADOOP\_PREFIX="/opt/hadoop-2.6.0/"

export PATH=$PATH:$HADOOP\_PREFIX/bin

export PATH=$PATH:$HADOOP\_PREFIX/sbin

export HADOOP\_COMMON\_HOME=${HADOOP\_PREFIX}

export HADOOP\_MAPRED\_HOME=${HADOOP\_PREFIX}

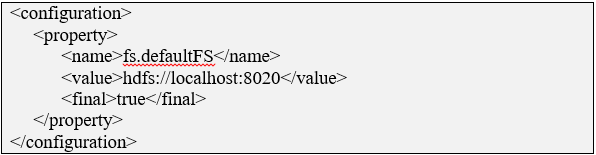
export HADOOP\_HDFS\_HOME=${HADOOP\_PREFIX}

export YARN\_HOME=${HADOOP\_PREFIX}

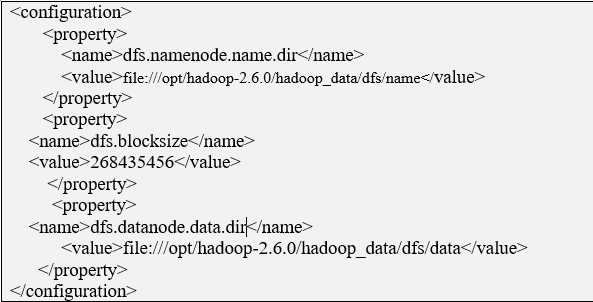
export JAVA\_HOME="/usr/lib/jvm/jdk1.8.0\_05"

export PATH=$PATH:$JAVA\_HOME/bin

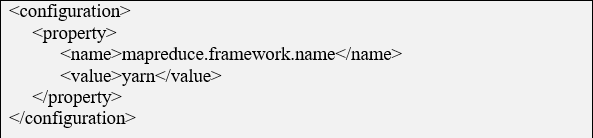
1. Save the file opened in nano (ctrl+x 🡪 y 🡪 enter) and run command:
   1. *source ~/.bash\_profile*
2. Go to /opt/hadoop-2.6.0/etc/hadoop directory and open **core-site.xml**. Enter following properties in it:



1. Go to /opt/hadoop-2.6.0/etc/hadoop directory and open **hdfs-site.xml**. Enter following properties in it:



1. Go to /opt/hadoop-2.6.0/etc/hadoop directory and open **mapred-site.xml**. Enter the following properties in it:



1. Go to /opt/hadoop-2.6.0/etc/hadoop directory and open **yarn-site.xml.** Enter the following properties in it:

|  |
| --- |
| <configuration>  <property>  <name>yarn.resourcemanager.address</name>  <value>localhost:8032</value>  </property>  <property>  <name>yarn.resourcemanager.scheduler.address</name>  <value>localhost:8030</value>  </property>  <property>  <name>yarn.resourcemanager.resource-tracker.address</name>  <value>localhost:8031</value>  </property>  <property>  <name>yarn.resourcemanager.admin.address</name>  <value>localhost:8033</value>  </property>  <property>  <name>yarn.resourcemanager.webapp.address</name>  <value>localhost:8088</value>  </property>  <property>  <name>yarn.resourcemanager.scheduler.class</name>  <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</value>  </property>  <property>                <name>yarn.nodemanager.aux-services</name>                <value>mapreduce\_shuffle</value>         </property>         <property>                 <name>yarn.nodemanager.local-dirs</name>                 <value>file:///opt/hadoop-2.6.0/hadoop\_data/yarn/yarn.nodemanager.local-dirs</value>         </property>         <property>                 <name>yarn.nodemanager.log-dirs</name>                 <value>file:///opt/hadoop-2.6.0/hadoop\_data/yarn/logs</value>         </property>         <property>              <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>              <value>org.apache.hadoop.mapred.ShuffleHandler</value>          </property>  </configuration> |

1. Go to /opt/hadoop-2.6.0/etc/hadoop directory and open or create **hadoop-env.sh**. enter/change java path to your path of java installation :  
              export JAVA\_HOME=/usr/lib/jvm/jdk1.8.0\_05

     You can check your java installation path by command:

$*echo $JAVA\_HOME*

Open **~/.bashrc** file and write the code

Export JAVA\_HOME = /usr/lib/jvm/jdk1.8.0\_05

Export HADOOP\_HOME=/opt/hadoop-2.6.0

Export PATH= $JAVA\_HOME/bin : $HADOOP\_HOME/bin :$PATH

Then **source ~/.bashrc** for refresh

1. Last step before running the service up is to format the namenode, run the command from hadoop folder:  
                    $*hadoop namenode –format*
2. After successful format of namenode, run the hadoop services as:

To start all services **start-all.sh**

|  |  |
| --- | --- |
| **Service** | **Command** |
| Namenode | hadoop-daemon.sh start namenode |
| Datanode | hadoop-daemon.sh start datanode |
| Resourcemanager | yarn-daemon.sh start resourcemanager |
| Nodemanager | yarn-daemon.sh start nodemanager |
| Job History Server | mr-jobhistory-daemon.sh start historyserver |

1. Logs could be accessible at /opt/hadoop-2.6.0/logs folder.

* Follow the steps below to install 2.x on a **multi node (cluster)** Linux machine:

     1.   Create a folder for Hadoop in “opt” say /opt/hadoop-2.6.0-dist

2.  If you do not have permission for creating folder in opt, you could execute following

          Command to change the mode of “opt” folder

   $ sudo chmod -R xxx /opt/ (here xxx could be the Linux permissions set like 777).

      3. $cd /opt/hadoop-2.6.0-dist

     4.  Download 2.x version tar.gz file of Hadoop from [here](http://www.bizdirusa.com/mirrors/apache/hadoop/common/)

     5. Untar the file to a folder /opt/hadoop-2.6.0-dist using command:  
 $tar -xzvf hadoop-filename.tar.gz

     6.  Move hadoop setup folder to /opt/

   $mv hadoop-2.6.0 /opt/

     7. $cd /opt/hadoop-2.6.0

**8.** Before installing hadoop, configure the environment variables:  
   Open or create .bash\_profile in home folder of user:   
   $ sudo nano ~/.bash\_profile

Enter the following env. variables in this file:

export HADOOP\_PREFIX="/opt/hadoop-2.6.0/"

export PATH=$PATH:$HADOOP\_PREFIX/bin

export PATH=$PATH:$HADOOP\_PREFIX/sbin

export HADOOP\_COMMON\_HOME=${HADOOP\_PREFIX}

export HADOOP\_MAPRED\_HOME=${HADOOP\_PREFIX}

export HADOOP\_HDFS\_HOME=${HADOOP\_PREFIX}

export YARN\_HOME=${HADOOP\_PREFIX}

export JAVA\_HOME="/usr/lib/jvm/jdk1.8.0\_05"

export PATH=$PATH:$JAVA\_HOME/bin

  9. Save the file and run the command:   $source ~/.bash\_profile

     10. Go to /opt/hadoop-2.6.0/etc/hadoop directory and open **core-site.xml**. Enter following

          properties:

|  |
| --- |
| <configuration>  <property>  <name>fs.defaultFS</name>  <value>hdfs://hadoopnode1:8020</value>  <final>true</final>  </property>  <property>  <name>io.file.buffer.size</name>  <value>131072</value>  </property>  </configuration> |

     11. Go to /opt/hadoop-2.6.0/etc/hadoop directory and open **hdfs-site.xml**. Enter following

         Properties:

|  |
| --- |
| <configuration>  <property>  <name>dfs.namenode.name.dir</name>  <value>file:///opt/hadoop-2.6.0/hadoop\_data/dfs/name</value>  </property>  <property>  <name>dfs.blocksize</name>  <value>268435456</value>  </property>  <property>  <name>dfs.namenode.handler.count</name>  <value>100</value>  </property>  <property>  <name>dfs.datanode.data.dir</name>  <value>file:///opt/hadoop-2.6.0/hadoop\_data/dfs/data</value>  </property>  </configuration> |

      12. Go to /opt/hadoop-2.6.0/etc/hadoop directory and open **mapred-site.xml**. Enter

             following properties in it:

|  |
| --- |
| <configuration>  <property>  <name>mapreduce.framework.name</name>  <value>yarn</value>  </property>   </configuration> |

      13. Go to /opt/hadoop-2.6.0/etc/hadoop directory and open **yarn-site.xml**. Enter following

           properties:

|  |
| --- |
| <configuration>  <property>  <name>yarn.resourcemanager.address</name>  <value>hadoopnode1:8032</value>  </property>  <property>  <name>yarn.resourcemanager.scheduler.address</name>  <value>hadoopnode1:8030</value>  </property>  <property>  <name>yarn.resourcemanager.resource-tracker.address</name>  <value>hadoopnode1:8031</value>  </property>  <property>  <name>yarn.resourcemanager.admin.address</name>  <value>hadoopnode1:8033</value>  </property>  <property>  <name>yarn.resourcemanager.webapp.address</name>  <value>hadoopnode1:8088</value>  </property>  <property>  <name>yarn.resourcemanager.scheduler.class</name>                   <value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</value>  </property>          <property>                <name>yarn.nodemanager.aux-services</name>                <value>mapreduce\_shuffle</value>         </property>         <property>                 <name>yarn.nodemanager.local-dirs</name>                 <value>file:///opt/hadoop-2.6.0/hadoop\_data/yarn/yarn.nodemanager.local-dirs</value>         </property>         <property>                 <name>yarn.nodemanager.log-dirs</name>                 <value>file:///opt/hadoop-2.6.0/hadoop\_data/yarn/logs</value>         </property>         <property>              <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>              <value>org.apache.hadoop.mapred.ShuffleHandler</value>          </property>  </configuration> |

Note: here hadoopnode1 is the hostname of host that would run Resource Manager

      14. Go to /opt/hadoop-2.6.0/etc/hadoop directory and open or create **hadoop-env.sh**.

            enter/change java path to your path of java installation :

                  export JAVA\_HOME=/usr/lib/jvm/jdk1.8.0\_05

       15. Edit slaves file and add the worker nodes (the nodes that would run datanode and

             nodemanager) name or ip address. A single node can act as a Master and Worker node both, but there is only one node in cluster that can act as Master.

**$ nano /opt/hadoop-2.6.0/etc/hadoop/slaves**

        Add Node lists as a new line separated entries:

hadoopnode1

hadoopnode2

**Cloning here of machine then following cloning steps**

16. Enable passwordless Ssh between the Master Node (the node running Resource

             Manager and Worker Nodes by running following commands on Master Node for each

             Worker node as host:

a. ssh-keygen

b. ssh-copy-id -i localhost

$ ssh-agent $SHELL

$ ssh-add

$ ssh-copy-id -i <hadoopnode2>

       Ensure the passwordless ssh is working fine by doing ssh to some of the nodes as   
       $ssh user@<hadoopnode1>

        17. Copy the setup on Master node to all the Worker nodes as:

         $scp -r /opt/hadoop-2.6.0 hadoop@hadoopnodename:/opt/

         $scp ~/.bash\_profile hadosuop@hadoopnodename:~/

        18. Edit /etc/hosts of all nodes to add mapping of node’s hostname with the ip address for

All nodes in the cluster.

#127.0.0.1 localhost

#127.0.1.1 hadoopnode1

192.168.56.101 hadoopnode1

192.168.56.102hadoopnode2

        19. Last step before running the service up is to format the namenode, run the command

         $hdfs namenode –format

         20. After successful format of namenode, run the hadoop services as:

$start-all.sh

         21. Logs could be accessible at /opt/hadoop-2.6.0/logs folder.

**CHAPTER 3**

**COMPONENTS OF HADOOP**

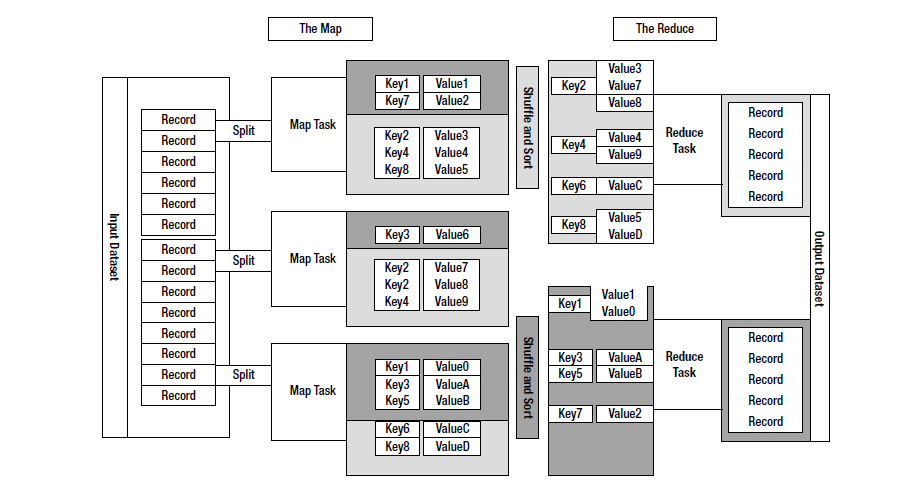
Hadoop has two major components:

* **Map Reduce**
* **HDFS (Hadoop Distributed File System)**

The key technology for Hadoop is the Map Reduce programming model and Hadoop Distributed File System. The operation on large data is not possible in serial programming paradigm. Map Reduce do task parallel to accomplish work in less time which is the main aim of this technology. Map Reduce require special file system. In the real scenario, the data which are in terms on petabyte. To store and maintain this much data on distributed commodity hardware, Hadoop Distributed File System is invented. It is basically inspired by Google File System.

## 3.1 Map Reduce:

Map Reduce is a framework for processing highly distributable problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes use the same hardware) or a grid (if the nodes use different hardware). Computational processing can occur on data stored either in a file system (unstructured) or in a database (structured).



**Figure 3.1 Map Reduce Programming Model**

**"Map" step:** The master node takes the input, partitions it up into smaller sub-problems, and distributes them to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node.

**"Reduce" step:** The master node then collects the answers to all the sub-problems and combines them in some way to form the output – the answer to the problem it was originally trying to solve.

Map Reduce allows for distributed processing of the map and reduction operations. Provided each mapping operation is independent of the others, all maps can be performed in parallel – though in practice it is limited by the number of independent data sources and/or the number of CPUs near each source. Similarly, a set of 'reducers' can perform the reduction phase - provided all outputs of the map operation that share the same key are presented to the same reducer at the same time. While this process can often appear inefficient compared to algorithms that are more sequential, Map Reduce can be applied to significantly larger datasets than "commodity" servers can handle – a large server farm can use Map Reduce to sort a petabyte of data in only a few hours. The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one mapper or reducer fails, the work can be rescheduled – assuming the input data is still available.

Map Reduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model.

**3.1.1 Map Reduce Programming Model:**

The computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the Map Reduce library expresses the computation as two functions: Map and Reduce. Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The Map Reduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function. The Reduce function, also written by the user, accepts an intermediate key I and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory.

**3.1.1.1 MAP:**

**map (in\_key, in\_value) -> (out\_key, intermediate\_value) list**

**Figure 3.2 Map Technology**

**Example: Upper-case Mapper**

let map(k, v) = emit(k.toUpper(), v.toUpper())

(“foo”, “bar”) --> (“FOO”, “BAR”)

(“Foo”, “other”) -->(“FOO”, “OTHER”)

(“key2”, “data”) --> (“KEY2”, “DATA”)

**3.1.1.2 REDUCE:**

**reduce (out\_key, intermediate\_value list) ->out\_value list**

**Figure 3.3 Reducing Technology**

**Example: Sum Reducer**

let reduce(k, vals)

sum = 0

foreachint v in vals:

sum += v

emit(k, sum)

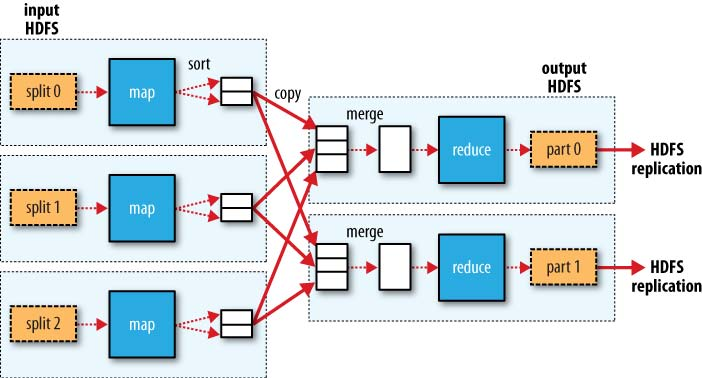
(“A”, [42, 100, 312]) --> (“A”, 454)

(“B”, [12, 6, -2]) --> (“B”, 16)

Hadoop Map-Reduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A Map-Reduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

A Map Reduce job is a unit of work that the client wants to be performed: it consists of the input data, the Map Reduce program, and configuration information. Hadoop runs the job by dividing it into tasks, of which there are two types: map tasks and reduce tasks. There are two types of nodes that control the job execution process: a job tracker and a number of task trackers. The job tracker coordinates all the jobs run on the system by scheduling tasks to run on task trackers.



**Figure 3.4 Hadoop Map Reduce**

**Example of Word Count:**

****

### Figure 3.5 Driver Code

### 

### Figure 3.6 Mapper Class

### 

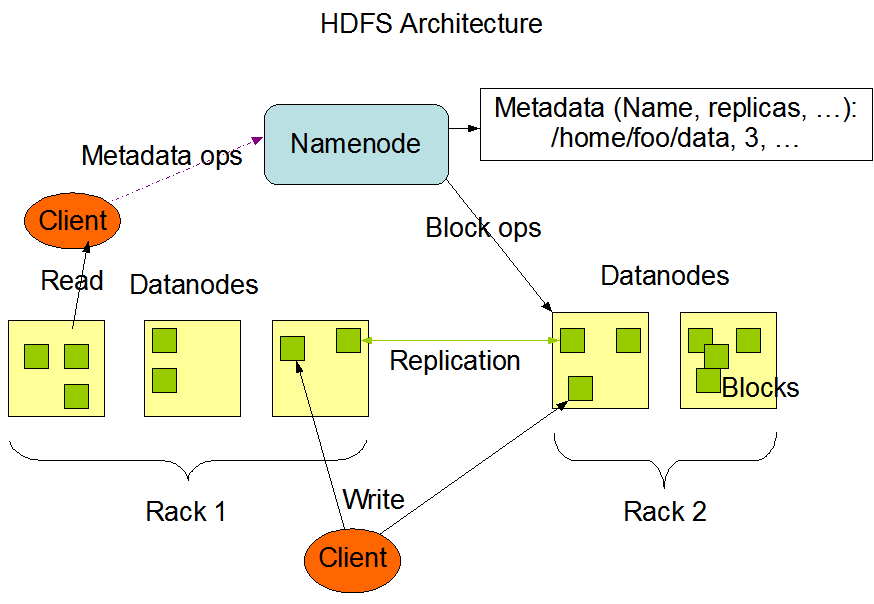
### Figure 3.7 Reducer Class

### 

### Figure 3.8 Output of Word Count

### 3.2 HDFS (HADOOP DISTRIBUTED FILE SYSTEM):

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 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 Nutch web search engine project. HDFS is now an Apache Hadoop subproject.



**Figure 3.9 HDFS Architecture**

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system’s clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

The NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software. Each of the other machines in the cluster runs one instance of the DataNode software. The architecture does not preclude running multiple DataNodes on the same machine but in a real deployment that is rarely the case.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

Filesystems that manage the storage across a network of machines are called distributed filesystems. Since they are network-based, all the complications of network programming kick in, thus making distributed filesystems more complex than regular disk filesystems. For example, one of the biggest challenges is making the filesystem tolerate node failure without suffering data loss. Hadoop comes with a distributed filesystem called HDFS, which stands for HadoopDistributed Filesystem.

**HDFS, the Hadoop Distributed File System, is a distributed file system designed to hold very large amounts of data (terabytes or even petabytes), and provide high-throughput access to this information.** Files are stored in a redundant fashion across multiple machines to ensure their durability to failure and high availability to very parallel applications.

## 3.3 HDFS CONCEPTS

### a) Blocks

A disk has a block size, which is the minimum amount of data that it can read or write. Filesystems for a single disk build on this by dealing with data in blocks, which are an integral multiple of the disk block size. Filesystem blocks are typically a few kilobytes in size, while disk blocks are normally 512 bytes. This is generally transparent to the filesystem user who is simply reading or writing a file—of whatever length. However, there are tools to do with filesystem maintenance, such as dfand fsck, that operate on the filesystem block level. HDFS too has the concept of a block, but it is a much larger unit—64 MB by default. Like in a filesystem for a single disk, files in HDFS are broken into block-sized chunks, which are stored as independent units. Unlike a filesystem for a single disk, a file in HDFS that is smaller than a single block does not occupy a full block’s worth of underlying storage. When unqualified, the term “block” in this book refers to a block in HDFS.

### b) Namenodes and Datanodes

A HDFS cluster has two types of node operating in a master-worker pattern: a namenode(the master) and a number of datanodes(workers). The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files: the namespace image and the edit log. The namenode also knows the datanodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from datanodes when the system starts. A client accesses the filesystem on behalf of the user by communicating with the namenode and datanodes.

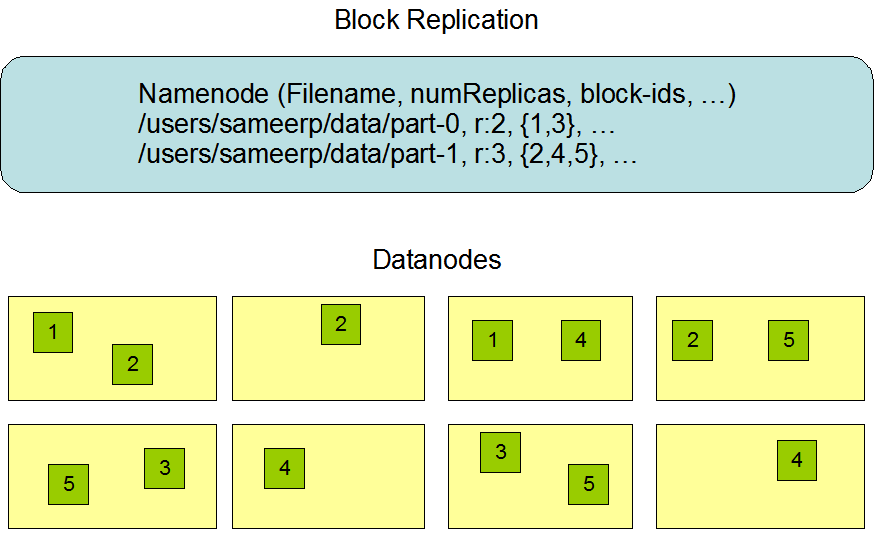
The client presents a POSIX-like filesystem interface, so the user code does not need to know about the namenode and datanode to function. Datanodes are the work horses of the filesystem. They store and retrieve blocks when they are told to (by clients or the namenode), and they report back to the namenode periodically with lists of blocks that they are storing. Without the namenode, the filesystem cannot be used. In fact, if the machine running the namenode were obliterated, all the files on the filesystem would be lost since there would be no way of knowing how to reconstruct the files from the blocks on the datanodes. For this reason, it is important to make the namenode resilient to failure.

**c) The File System Namespace**

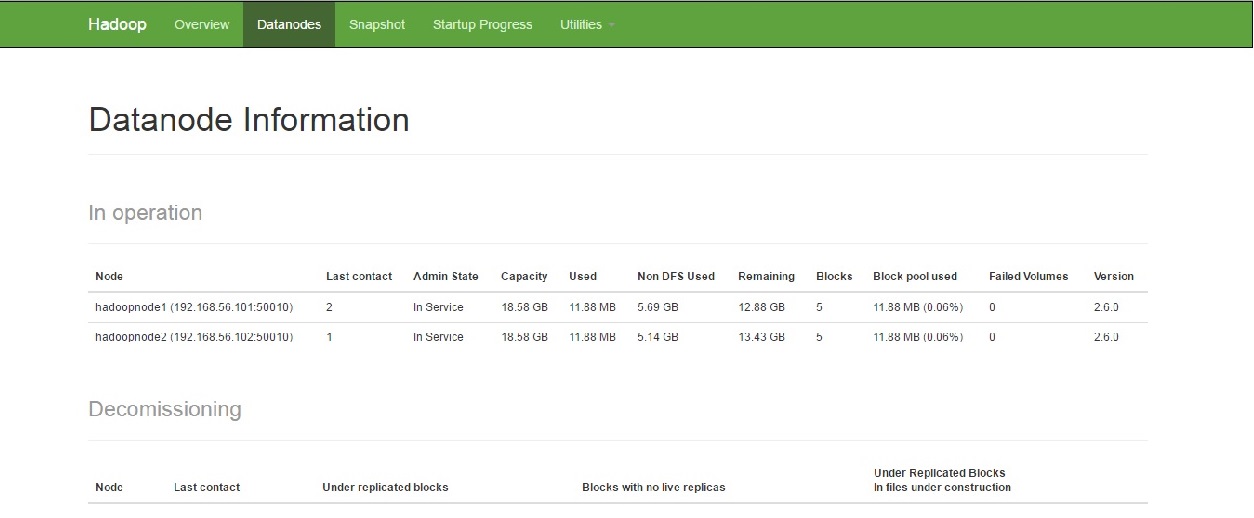
HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories. The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file. HDFS does not yet implement user quotas or access permissions. HDFS does not support hard links or soft links. However, the HDFS architecture does not preclude implementing these features.

### d) Data Replication

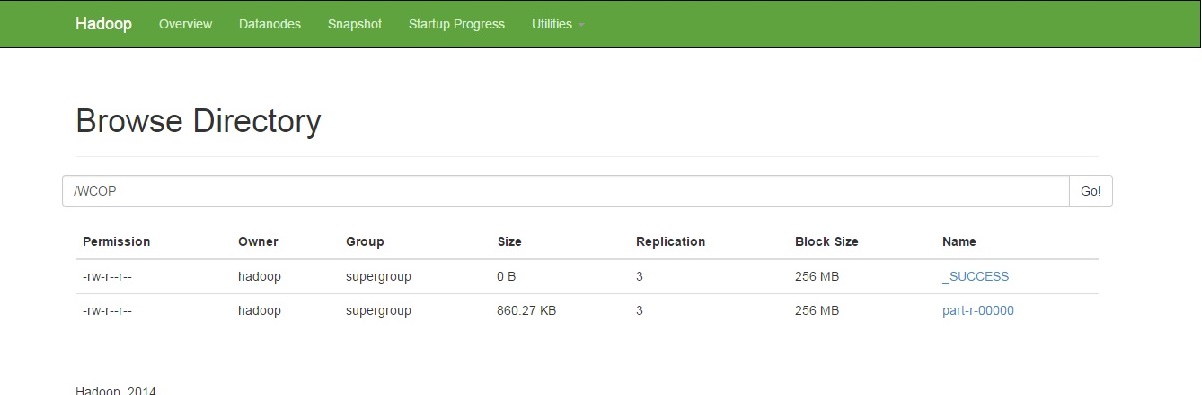
HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later.



**Figure 3.10 Data replication**



**Figure 3.11 Data Nodes**



**Figure 3.12 Browse Files**

**3.4 HADOOP FILESYSTEMS**

Hadoop has an abstract notion of filesystem, of which HDFS is just one implementation. The Java abstract class org.apache.hadoop.fs.FileSystem represents a filesystem in Hadoop, and there are several concrete implementations, which are described in following table.

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Type** | **Command** | **Description** |
| **Local** | file | fs.LocalFileSystem | A filesystem for a locally connected disk with client-side checksums.Use RawLocalFileSystem for a local filesystem with no checksums. |
| **HDFS** | hdfs | hdfs.DistributedFileSystem | Hadoop’s distributed filesystem. HDFS is designed to work efficiently in conjunction with Map-Reduce. |
| **HAR** | har | Fs.HarFileSystem | A filesystem layered on another filesystem for archiving files. Hadoop Archives are typically used for archiving files in HDFS to reduce the namenode’s memory usage |
| **KFS(Cloud Store)** | Kfs | fs.kfs.KosmosFileSystem | CloudStore (formerly Kosmosfilesystem) is a distributed filesystem like HDFS or Google’s GFS, written in C++. |
| **FTP** | ftp | fs.ftp.FtpFileSystem | A filesystem backed by an FTP server. |

**Table: 3.1 Various HadoopFilesystems**

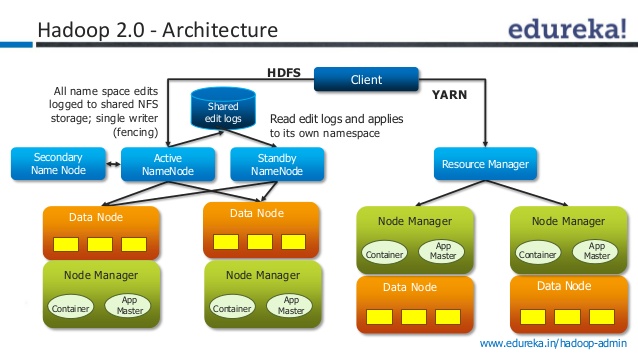
**3.5 HADOOP ARCHIVES**

HDFS stores small files inefficiently, since each file is stored in a block, and block metadata is held in memory by the namenode. Thus, a large number of small files can eat up a lot of memory on the namenode. (Note, however, that small files do not take up any more disk space than is required to store the raw contents of the file. For example, a 1 MB file stored with a block size of 128 MB uses 1 MB of disk space, not 128 MB.) Hadoop Archives, or HAR files, are a file archiving facility that packs files into HDFS blocks more efficiently, thereby reducing namenode memory usage while still allowing transparent access to files. In particular, Hadoop Archives can be used as input to MapReduce.

**CHAPTER 4**

**HADOOP ECOSYSTEM**

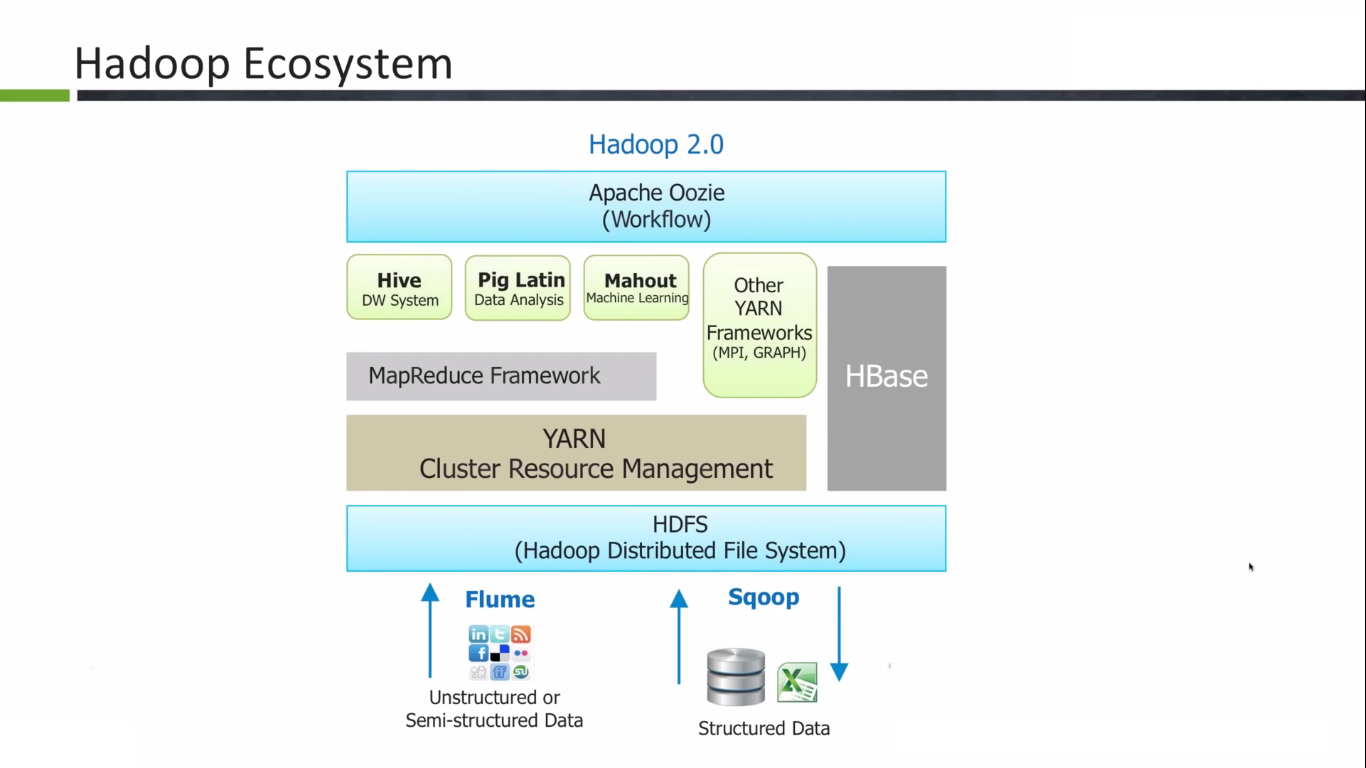
Apache Hadoop YARN (Yet another Resource Negotiator) is a cluster management technology. YARN is one of the key features in the second-generation Hadoop 2 version of the Apache Software Foundation's open source distributed processing framework. Originally described by Apache as a redesigned resource manager, YARN is now characterized as a large-scale, distributed operating system for big data applications.



**Figure 4.1 Hadoop 2.0 Architecture**

 In 2012, YARN became a sub-project of the larger Apache Hadoop project. Sometimes called MapReduce 2.0, YARN is a software rewrite that decouples MapReduce's resource management and scheduling capabilities from the data processing component, enabling Hadoop to support more varied processing approaches and a broader array of applications. For example, Hadoop clusters can now run interactive querying and streaming data applications simultaneously with MapReduce batch jobs. The original incarnation of Hadoop closely paired the Hadoop Distributed File System (HDFS) with the batch-oriented MapReduce programming framework, which handles resource management and job scheduling on Hadoop systems and supports the parsing and condensing of data sets in parallel.

YARN combines a central resource manager that reconciles the way applications use Hadoop system resources with node manager agents that monitor the processing operations of individual cluster nodes.  Running on commodity hardware clusters, Hadoop has attracted particular interest as a staging area and data store for large volumes of structured and unstructured data intended for use in analytics applications. Separating HDFS from MapReduce with YARN makes the Hadoop environment more suitable for operational applications that can't wait for batch jobs to finish.



**Figure 4.2 Hadoop Ecosystem**

**A) HBase** 

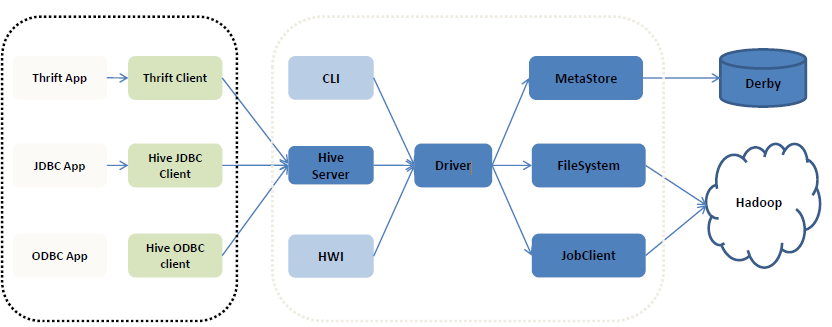
Hadoop Database or HBASE is a non-relational (NoSQL) database that runs on top of HDFS. HBASE was created for large table which have billions of rows and millions of columns with fault tolerance capability and horizontal scalability and based on Google Big Table. Hadoop can perform only batch processing, and data will be accessed only in a sequential manner for random access of huge data HBASE is used.



**Figure 4.3 Hbase Architecture**

**B) Hive** 

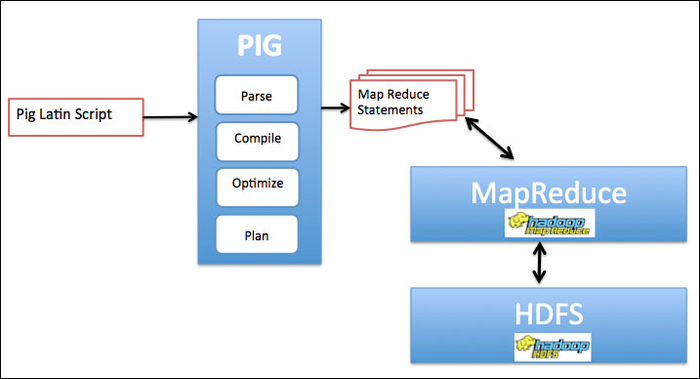
Many programmers and analyst are more comfortable with Structured Query Language than Java or any other programming language for which Hive is created by Facebook and later donated to Apache foundation. Hive mainly deals with structured data which is stored in HDFS with a Query Language similar to SQL and known as HQL (Hive Query Language). Hive also run Map reduce program in a backend to process data in HDFS but here programmer has not worry about that backend MapReduce job it will look similar to SQL and result will be displayed on console.

****

**Figure 4.4** Hive Architecture

**C) Pig** 

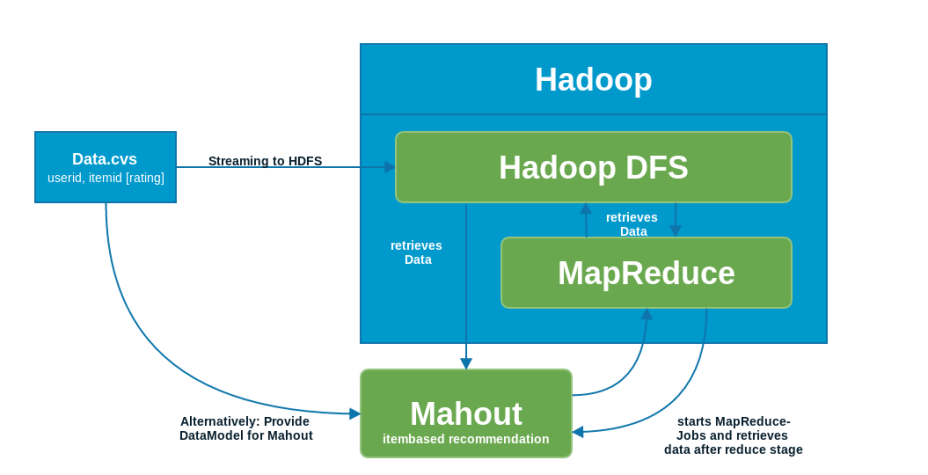
Similar to HIVE, PIG also deals with structured data using PIG LATIN language. PIG was originally developed at Yahoo to answer similar need to HIVE. It is an alternative provided to programmer who loves scripting and don't want to use Java/Python or SQL to process data. A Pig Latin program is made up of a series of operations, or transformations, that are applied to the input data which runs MapReduce program in backend to produce output.



**Figure 4.5 Pig architecture**

**D) Mahout**

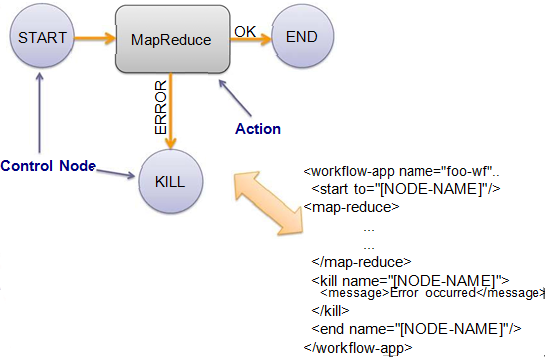
Mahout is an open source machine learning library from Apache written in java. The algorithms it implements fall under the broad umbrella of machine learning or collective intelligence. This can mean many things, but at the moment for Mahout it means primarily recommender engines (collaborative filtering), clustering, and classification. Mahout aims to be the machine learning tool of choice when the collection of data to be processed is very large, perhaps far too large for a single machine. In its current incarnation, these scalable machine learning implementations in Mahout are written in Java, and some portions are built upon Apache's Hadoop distributed computation project.



**Figure 4.6 Mahout Architecture**

**E) Oozie**

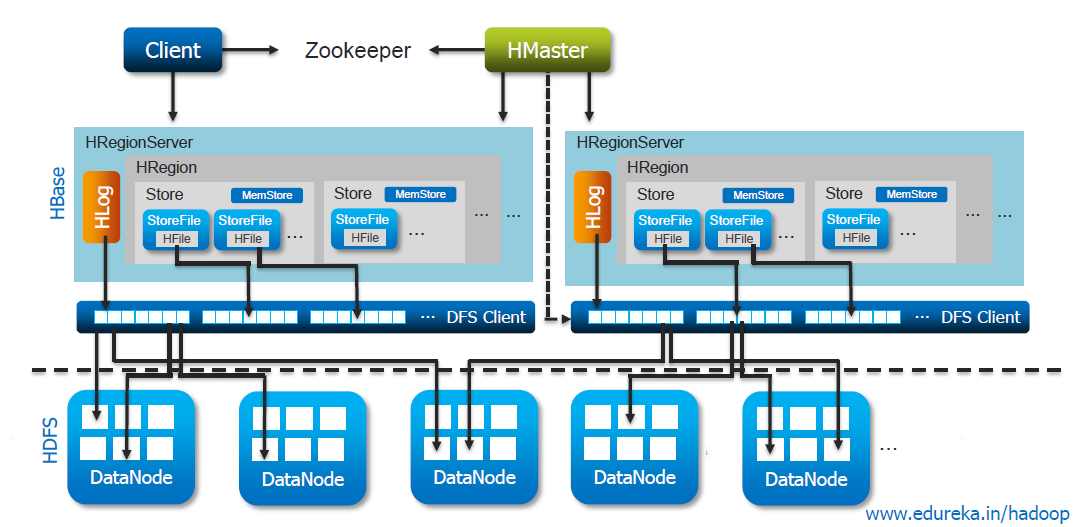
It is a workflow scheduler system to manage hadoop jobs. It is a server-based Workflow Engine specialized in running workflow jobs with actions that run Hadoop MapReduce and Pig jobs. Oozie is implemented as a Java Web-Application that runs in a Java Servlet-Container. Hadoop basically deals with bigdata and when some programmer wants to run many job in a sequential manner like output of job A will be input to Job B and similarly output of job B is input to job C and final output will be output of job C. To automate this sequence we need a workflow and to execute same we need engine for which OOZIE is used.



**Figure 4.7 Oozie Architecture**

**F) Zookeeper** 

Writing distributed applications is difficult because of partial failure may occur between nodes to overcome this Apache Zookeeper has been developed by maintaining an open-source server which enables highly reliable distributed coordination. ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. In case of any partial failure clients can connect to any node and be assured that they will receive the correct, up-to-date information.

****

**Figure 4.8 Zookeeper Architecture**

**F) Sqoop**

This tool is designed for efficiently transferring bulk data between Apache Hadoop and structured datastores (RDBMS).

Bulk data transfer tool

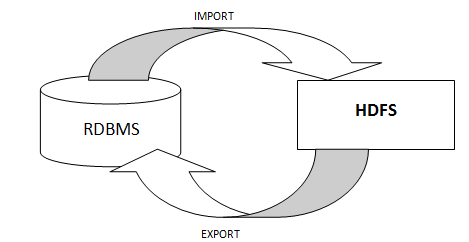
•To import and export data from a relational database into Hadoop for processing

•Map only job.

•command-line tool

•Integrates with Hive & Hbase

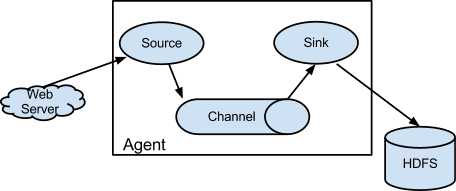
•Support plugins via connector based architecture



**Figure 4.9 Sqoop Architecture**

**G) Flume**

Flume is an open source programming undertaking which is made by cloudera to go about as an organization for gathering and moving enormous measure of data around a Hadoop bundle as data is conveyed or in no time. Crucial use case of flume is to gather log records from all machines in cluster to continue on them in a united store, for instance, HDFS. In it, we have to make data streams by building up chains of sensible centre points and partner them to source and sink. For example, if you have to move data from an apache access sign into HDFS then you have to make a source by tail access.log and use an astute centre point to course this to a HDFS sink. By far most of flume game plans have three level diagrams. The administrator’s level has flume masters assembled with wellsprings of data which is to be moved. Power level involve various gatherers each of which accumulate data coming in from distinctive authorities and forward it on to limit level which include archive system like HDFS or GFS. A Flume agent is a JVM process which has 3 components -Flume Source, Flume Channel and Flume Sink through which events propagate after initiated at an external source

****

**Figure 4.10 Flume Architecture**

## 

## CHAPTER-5

## APPLICATION OF HADOOP

5.1.

Amazon S3 (Simple Storage Service) is a data storage service. You are billed monthly for storage and data transfer. Transfer between S3 and AmazonEC2 is free. This makes use of S3 attractive for Hadoop users who run clusters on EC2.

Hadoop provides two filesystems that use S3 :-

1. S3 Native FileSystem (URI scheme: s3n)

2. S3 Block FileSystem (URI scheme: s3)

5.2.

Facebook’s engineering team has posted some details on the tools it’s using to analyze the huge data sets it collects. One of the main tools it uses is Hadoop that makes it easier to analyze vast amounts of data.

Some interesting tidbits from the post:

Facebook has multiple Hadoop clusters deployed now - with the biggest having about 2500 cpu cores and 1 PetaByte of disk space. They are loading over 250 gigabytes of compressed data (over 2 terabytes uncompressed) into the Hadoop file system every day and have hundreds of jobs running each day against these data sets. The list of projects that are using this infrastructure has proliferated - from those generating mundane statistics about site usage, to others being used to fight spam and determine application quality.

5.3.

Yahoo! recently launched the world's largest Apache Hadoop production application. The Yahoo! Search Webmap is a Hadoop application that runs on a more than 10,000 core Linux cluster and produces data that is now used in every Yahoo! Web search query.

The Webmap build starts with every Web page ***crawled*** by Yahoo! and produce a database of all known Web pages and sites on the internet and a vast array of data about every page and site. This derived data feeds the Machine Learned Ranking algorithms at the heart of Yahoo! Search.

Some Webmap size data:Number of links between pages in the index: roughly 1 trillion links

Size of output: over 300 TB, compressed!

Number of cores used to run a single Map-Reduce job: over 10,000

Raw disk used in the production cluster: over 5 Petabytes

**CHAPTER 6**

**INFOSPHERE BIGINSIGHTS**

Queries and Output

* Top 5 Gainer of the stock data on 25-Nov-2009

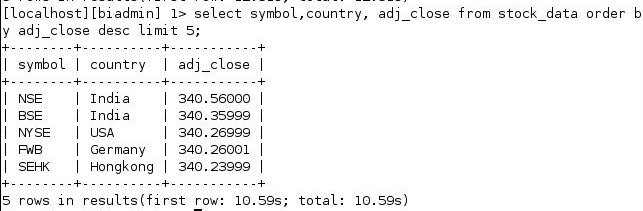
******

Fig 6.1: Query and Output for Top Gainer

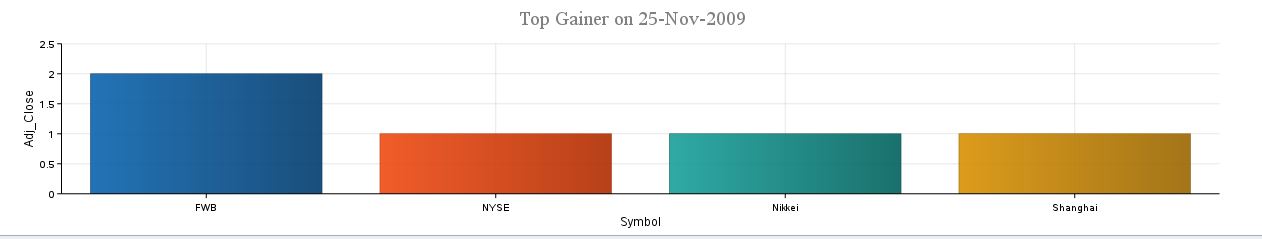
******

Fig 6.2: Bar Chart showing Top Gainer

* Count of stocks with sector HealthCare and Manufacturing

******

Fig 6.3: Query for stock count

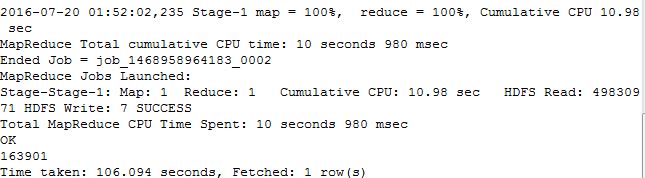
******

Fig 6.4: Output

* Asia Stock Exchange



Fig 6.5: Query for Asia Stock

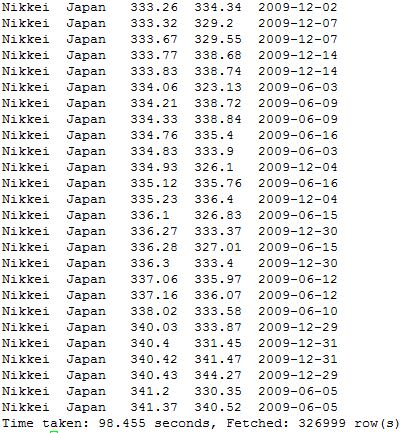
****

Fig 6.6: Output for Asia Stock

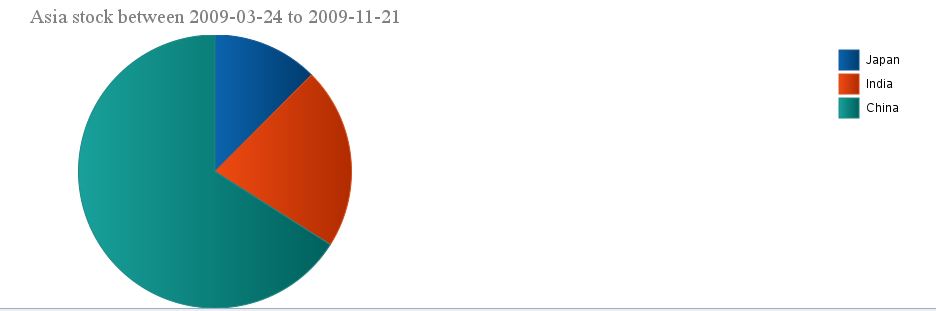


Fig 6.7: Pie Chart for Asia Stock Exchange

* Stock with Average of high value grouped by Symbol



Fig 6.8: Query for avg(high)



Fig 6.9: Output for Average High Value

* Stock of Germany with closing value between 15 and 21



Fig 6.10: Query for stock of Germany

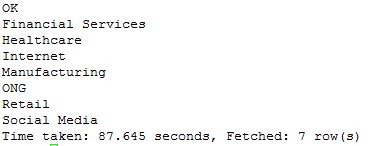


Fig 6.11: Output of Germany stock

* Partition of data based on Symbol



Fig 6.12: Query for Partition

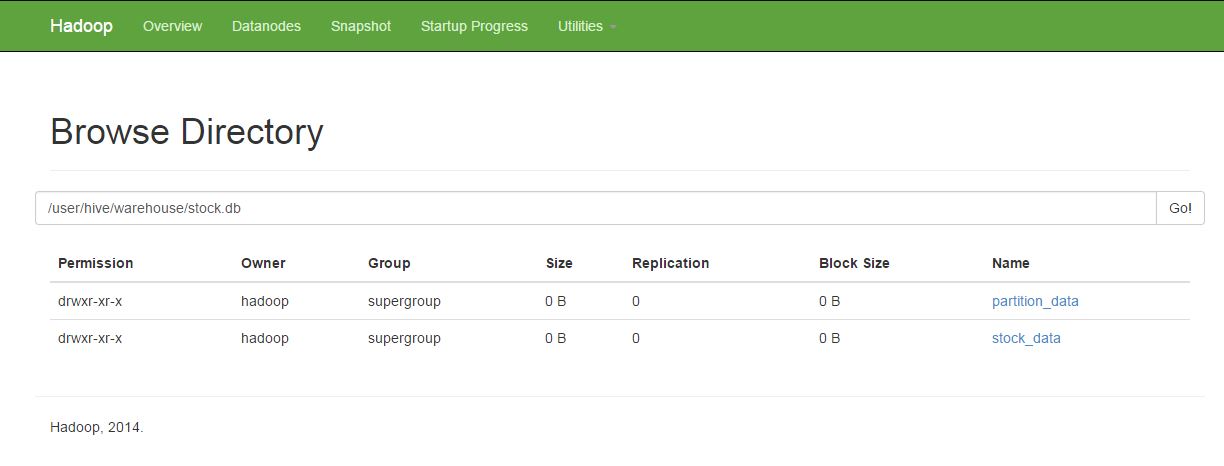


Fig 6.13: Output of Partition Table

* Stock data having open value greater than 16.



Fig 6.14: Query for open value greater than 16

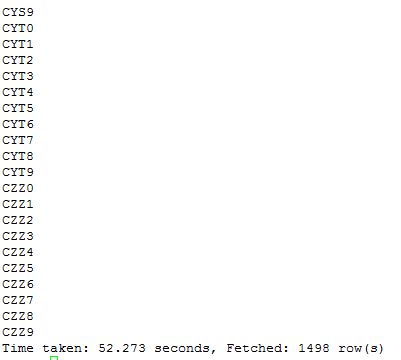


Fig 6.15: Output of open value

* Hong Kong stock data between 2009-03-24 to 2009-11-21



Fig 6.16: Query of Hong Kong stock

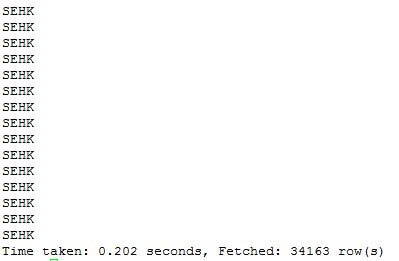
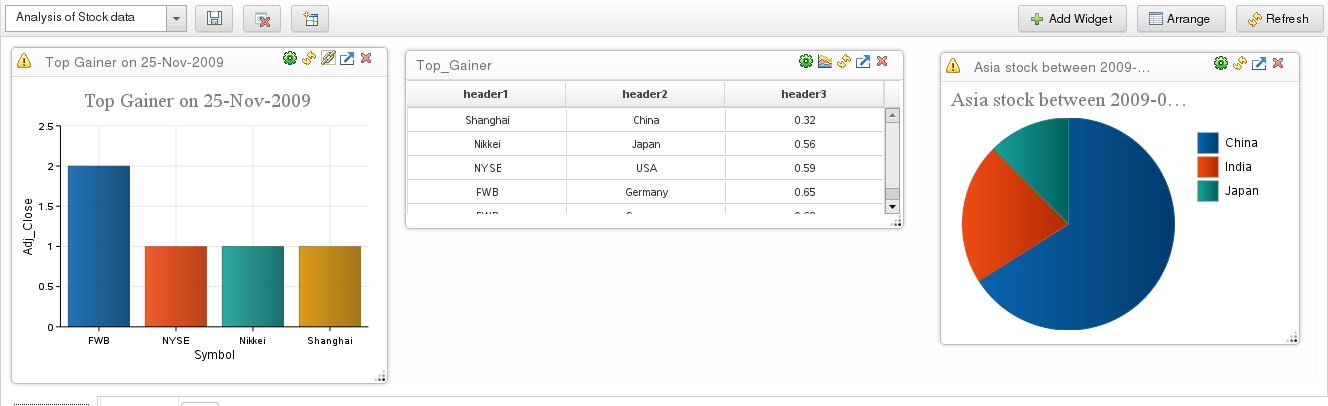
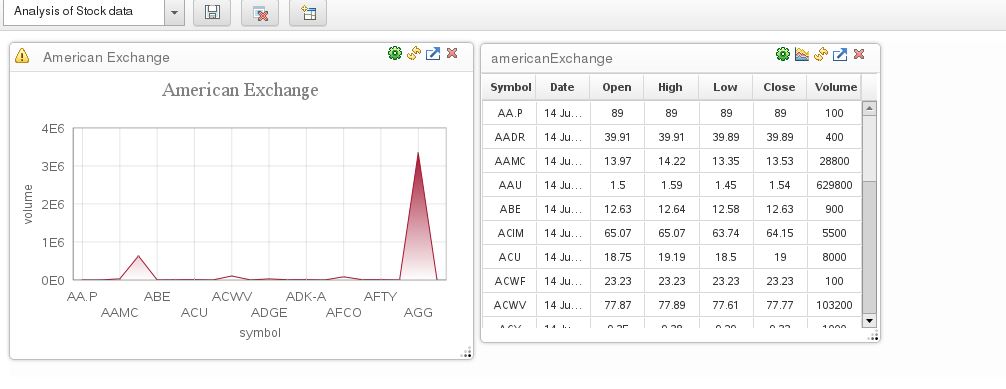


Fig 6.17: Output of Hong Kong stock

OVERALL ANALYSIS

(DASHBOARD)

******

******

CONCLUSION

* By the above description we can understand the need of Big Data in future, so Hadoop can be the best of maintenance and efficient implementation of large data.
* This technology has bright future scope because day by day need of data would increase and security issues also the major point. In nowadays many Multinational organizations prefer Hadoop over RDBMS.
* So major companies like facebook amazon,yahoo,linkedIn etc. are adapting Hadoop and in future there can be many names in the list.
* Hence Hadoop Technology is the best appropriate approach for handling the large data in smart way and its future is bright.

**REFERENCE**

1 .http://www.ibm.com/hadoop-training-thinking-at-scale

2. http://developer.yahoo.com/hadoop/tutorial/module1.html

3. http://hadoop.apache.org/core/docs/current/api/

4. http://hadoop.apache.org/core/version\_control.html

5http://wikipidea.in/apachehadoop.com