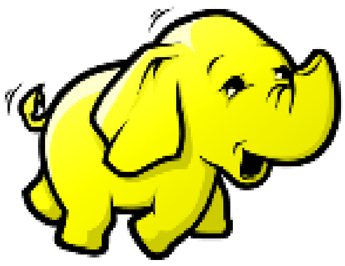
Wisdom Sprouts

We foster knowledge here

BIG DATA HADOOP

An Introduction



2015

WISDOM SPROUTS

Visit www.wisdomsprouts.com

CONTENTS

INTRODUCTION TO BIG DATA

Welcome to the age of Internet. Today if an Engineering Students fails in the exam, Facebook is the first one to know about it *(dude who will tell this to your parents)*. Today if a girl's Dog is not feeling well, Instagram is the first one to know about it *(instead go to a doctor babe)*. Shopping is just a matter of few clicks today. Next season of Kaun Banega Crorepati is supposed to launch a new life line Ping a Friend.

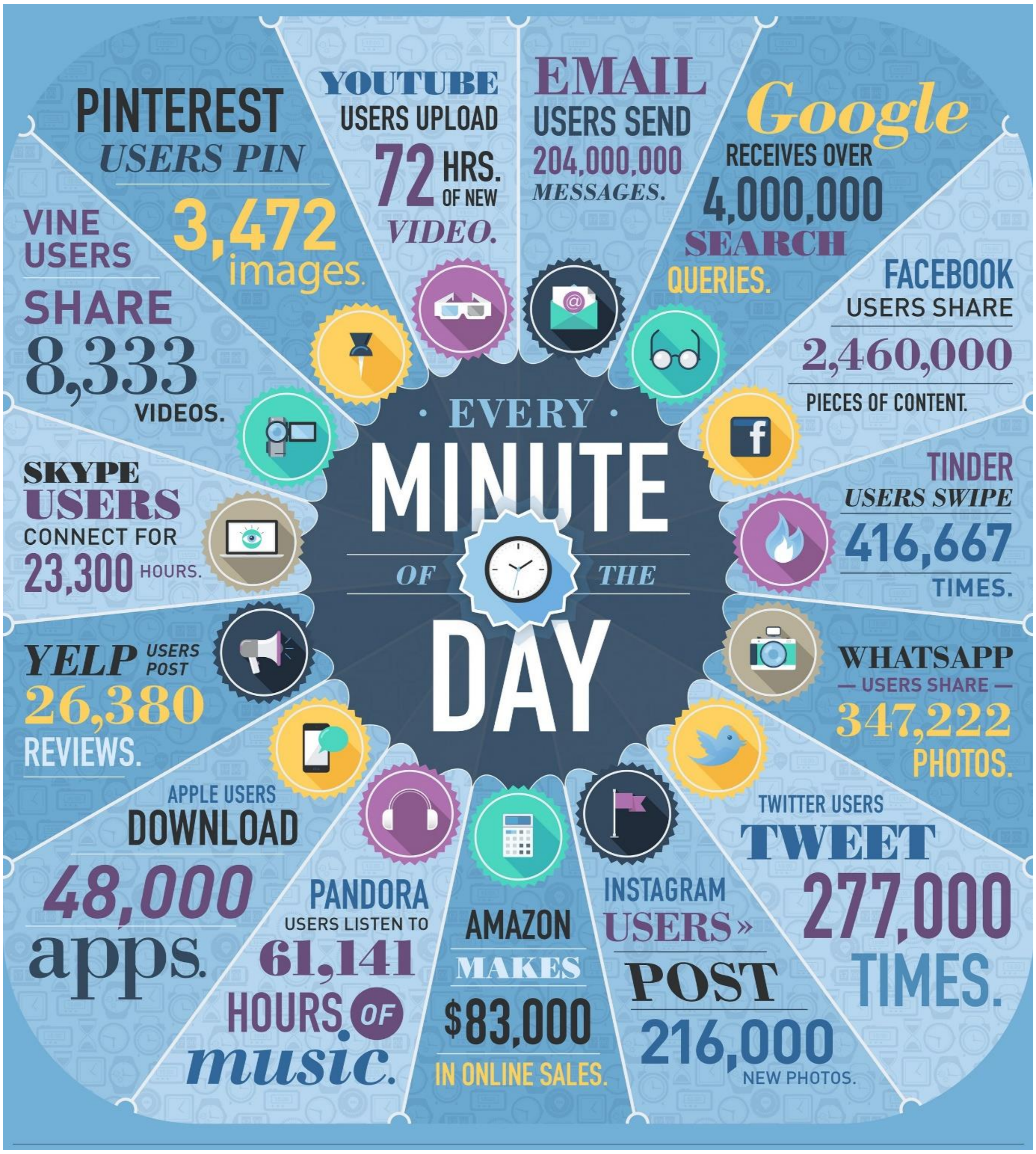

EVERY MINUTE OF THE DAY
PINTEREST USERS PIN 3,472 images.
YOUTUBE USERS UPLOAD 72 HRS. OF NEW VIDEO.
EMAIL USERS SEND 204,000,000 MESSAGES.
Google RECEIVES OVER 4,000,000 SEARCH QUERIES.
FACEBOOK USERS SHARE 2,460,000 PIECES OF CONTENT.
TINDER USERS SWIPE 416,667 TIMES.
WHATSAPP USERS SHARE 347,222 PHOTOS.
TWITTER USERS TWEET 277,000 TIMES.
INSTAGRAM USERS » POST 216,000 NEW PHOTOS.
AMAZON MAKES $83,000 IN ONLINE SALES.
PANDORA USERS LISTEN TO 61,141 HOURS OF music.
APPLE USERS DOWNLOAD 48,000 apps.
YELP USERS POST 26,380 REVIEWS.
SKYPE USERS CONNECT FOR 23,300 HOURS.
VINE USERS SHARE 8,333 VIDEOS.

Fig 1: Online Minute

In a minute millions of Tweets, Shares, Posts, Uploads, Transactions etc. are happening around us. But what is this? Ultimately this is Data. So what this has done is, massively increased the amount of data that gets collected. Sites like Facebook and Amazon are supposed to handle PBs of data every 30 Minutes. This data if compared may be more than what the entire database on the planet had in the year around 2007.

This data is collected and analysed by companies to find out the true sentiments of their Customers.

Companies feel that they have now got a platform to understand the Real Me of their Customers. Though bitter but this is the truth. If you get a Feedback form at the end of a Seminar, how would you normally fill it? Remember right? Akad Bakad Bambe Bo. And if there is a section called comments, then it is more like a one word answer – NOTHING!!!! So how can someone really get a true feedback out of this?

But what if the same thing was to be done online? If a friend of yours posts a picture of her, then you will search Google to get the best words for your comment with only intent that your comment should get more likes than her pic. The What's on your mind is always what's on Google's mind stole by your mind. Agree? Why this indiscrimination? Because nobody even mentions their names in the feedback forms. So who cares what we feel because nobody will ever find out that I have filled it *(But we forget that companies to whom this form belongs, they care for our answers)*. Same thing on FB!! Man all my friends and family is going to see my comment. So, it has to be the best. And this is why companies now believe that, if they analyse this online data in the proper way then can find a way to influence the behaviour of their customers.

HOW DOES THIS DATA LOOK LIKE?

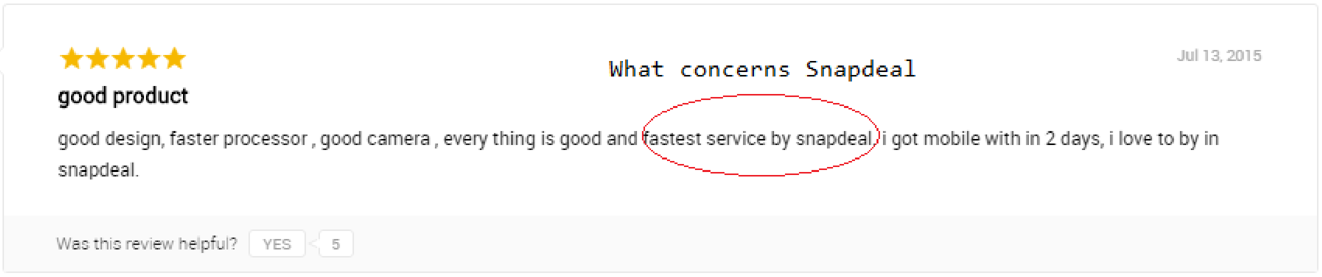
The online data that we are collecting over the past few years can be of any format. It can be Strings, Integers, Images, Videos, Audio, Banners, Graphs, etc. It can be anything and everything that you can ever imagine.

This data can be categorised as Structured Data *(data can be stored and processed in its true format. For example: Integers and Strings)* and Unstructured Data *(data that has to be converted into some convenient format for the purpose of storage and processing. For example: images converted to pixel values)*

WHAT IS DATA ANALYSIS?

The data we collect, is not always the exact stuff that we are looking for. It is always a mixture of Gold and Dirt.

For example: A feedback on Snapdeal looks like this:


★★★★★ What concerns Snapdeal Jul 13, 2015

good product

good design, faster processor , good camera , every thing is good and fastest service by snapdeal, i got mobile with in 2 days, i love to by in snapdeal.

Was this review helpful? YES 5

Fig 2: sample feedback

So fetching out the concerned details from the overall comment is a kind of data analysis.

Data Analysis is a big world in itself. It involves a lot of things like Data Mining, Data Processing, Predictive Analysis, Descriptive Analysis, Data Visualization etc.

HOW IS DATA ANALYSIS DONE?

Years of data is accumulated in large storage sectors and then using powerful tools and different algorithms, this data is analysed to figure out Business specific information about it. This does not only help you to understand the customer sentiments, but also helps you in doing a lot of stuff like understanding the trends, decision making, new launches etc.

But why is years of data needed for all this? We can do this by collecting some data as well, right? For example if I am an e-commerce site selling Poison, Rat Kill, and Sleeping pills *(Dumb Ways To Die)*, and if around 10 customer had visited me searching for Poison and finally 4 of them ended up buying Poison, 5 of them brought Sleeping pills and 1 of them brought Rat

Kill, then the 11th person that comes looking for Poison can be easily recommended the other two items, right? No!! If I do so, that person may charge me of encouraging suicidal tendencies and can sue me. Because he might be there to buy poison for some Lab experiments. But giving recommendation like sleeping pills will indicate that we want him to commit suicide. But what if I have been observing this trend from last 5 years that all those who came looking for poison on my site, have either committed a suicide or a murder the next day. In this case I can surely push up my recommendations. Right? So it's simple more the data you have, better can be your predictions.

WHERE IS THIS DATA STORED?

So we mean that we are analysing somewhere around 100s of 1000s of PBs of data every day. So store this much amount of data, surely we need strong and powerful servers that can scale limitlessly. So we make use of frameworks like Hadoop for handling this data.

WHAT HAPPENED TO THE TRADITIONAL SYSTEMS?

Our mothers are the smartest beings available on the planet. Do know why? Because they have the ability to sit on the last day of every month and plan the requirements for the coming month. Then the mother goes to the super market and gets all the stuff she wants and then she never has to buy anything extra in that month. Possibly the statistics of the previous months consumables or her experience might be giving her this ability to check on the things.

But what if one day suddenly around 15 people come to the house and decide to stay there forever. Now, these guys are not going to pay anything, neither are they going to help nor are they going to adjust on anything. She might have had still managed this number but all these people have different food habits and likes. The girls like fast food, boys like Chinese, males prefer south Indian items whereas the females prefer north Indian items. Apart from this the boys have a very weird habit. If they have a good dream the other night then the next day they will prefer eating like 6 times and if it was a nightmare then they may end up eating only once. So now the scenario stands in front of the mother that she has to cook for 15 people, the preparations differ and moreover she is not sure that how many times she might have to cook the food.

At this point of time, things go beyond her capacity and the most eligible person on the planet decides to quit and kicks off all those guests away.

Same thing happened with the data storage units that we were using traditionally and finally when things went beyond their capacity they proved to be inefficient.

WHY DID THIS HAPPEN?

CASE 1

How many types of shops are there?

1. Online Store (popular now)
2. Offline shops (popular earlier)

Right? If we try to briefly do a comparison between them on the basis of data that they store, this is what we may get

Products
Vendors
Invoices
Sizes, Price list
Customers
Inventory
Staff
Sales
Warehouse

Track your Order
Payment Gateway
Recommendations
User Reviews, Feedbacks, Ratings
Advertisements
Social Media Content

Products
Vendors
Invoices
Sizes, Price list
Customers
Inventory
Staff
Sales
Warehouse

Fig 3: Online vs. offline

The difference is pretty much clear. Not only the amount of data stored is multiplied but also if you compare the type of data stored then it has heavily migrated from structured data to unstructured data. Not only this but also the sources from where the data is collected has increased. Earlier it was only your databases from where you collected the information, but it can be somebody else's database as well (payment gateway, track your order, social media pages etc.). Was this situation imaginable before the advent of internet?

CASE 2

What will happen if one day, 500 customers visit a shop where the average daily number of customers is 50? The shopkeeper will panic and go crazy right? He won't be able to manage this. But online sites can handle this. Here's a small example

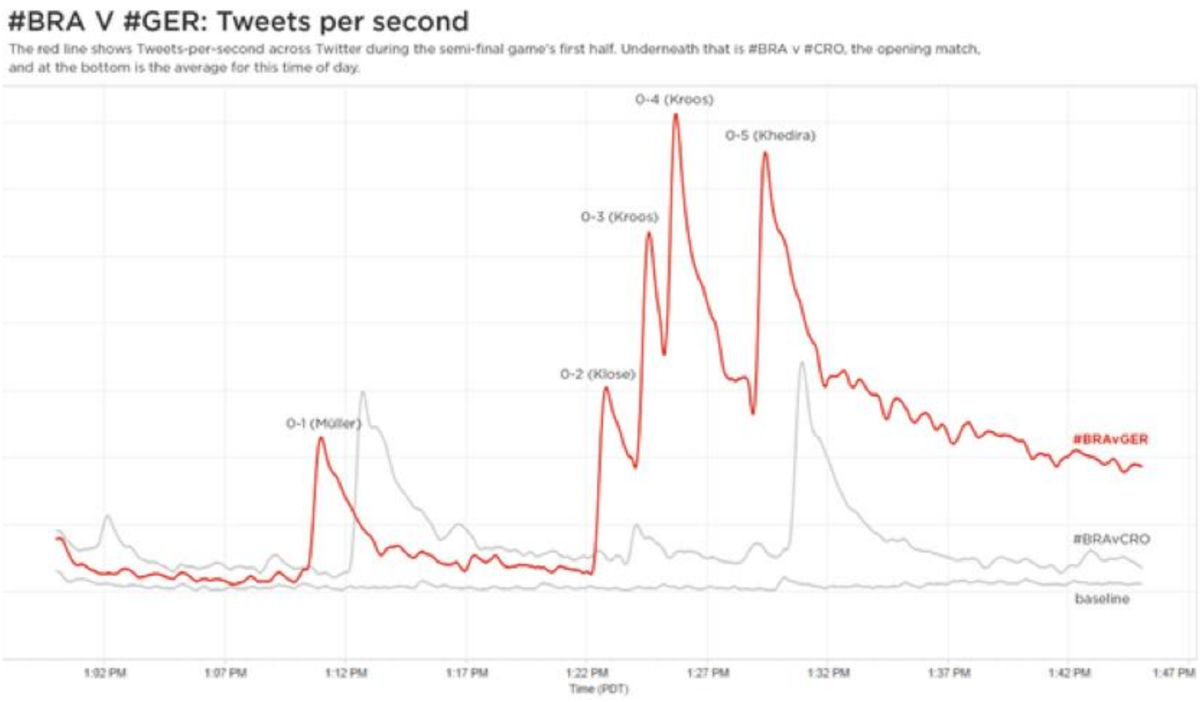
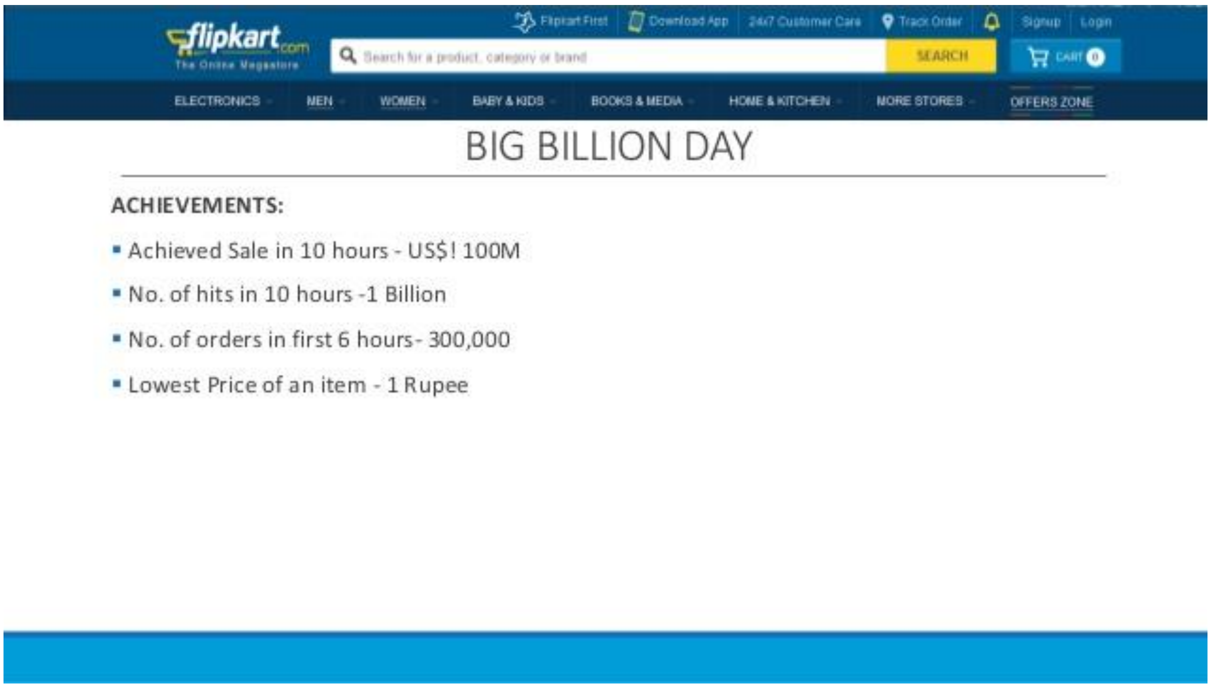


Fig 4: Twitter graph

Was this situation imaginable before the advent of internet?

CASE 3

One place, one shop, one day



BIG BILLION DAY

ACHIEVEMENTS:

- Achieved Sale in 10 hours - US$! 100M
- No. of hits in 10 hours -1 Billion
- No. of orders in first 6 hours- 300,000
- Lowest Price of an item - 1 Rupee

Fig 5: Flipkart Big Billion Day

Was this situation imaginable before the advent of internet?

All our 3 cases clearly suggest that all that happened was never imaginable before the advent of Internet. So how can we expect the storage entities built in an era when no one had the sight of internet, to be able to handle this situation?

SO THIS IS A PROBLEM RIGHT? WHAT IS IT CALLED?

Yes this is a problem of data that is huge, data that comes from a number of sources in a number of formats and that too in quick time. This is called as BIG DATA.

This being a problem can be defined in a number of ways, out of which some popular definitions are listed below

Big Data is the amount of data that is beyond the storage and the processing capabilities of a single physical machine.

Big Data is data whose scale, diversity, and complexity require new architecture, techniques, algorithms, and analytics to manage it and extract value and hidden knowledge from it.

Every day, we create 2.5 quintillion bytes of data — so much that 90% of the data in the world today has been created in the last two years alone. This data comes from everywhere: sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records, and cell phone GPS signals to name a few. This data is Big Data

CHARACTERISTICS OF BIG DATA

Big Data is generally defined as a composure of 3 Vs viz. Volume, Velocity and Variety. There are some other characteristics of this data as well. Let’s have a look at all of them.

1. VOLUME

The quantity of data that is generated is very important in this context. It is the size of the data which determines the value and potential of the data under consideration and whether it can actually be considered as Big Data or not. The name ‘Big Data’ itself contains a term which is related to size and hence the characteristic.

2. VARIETY

The next aspect of Big Data is its variety. This means that the category to which Big Data belongs to is also a very essential fact that needs to be known by the data analysts. This helps the people, who are closely analysing the data and are associated with it, to effectively use the data to their advantage and thus upholding the importance of the Big Data.

3. VELOCITY

The term ‘velocity’ in the context refers to the speed of generation of data or how fast the data is generated and processed to meet the demands and the challenges which lie ahead in the path of growth and development.

4. VARIABILITY

This is a factor which can be a problem for those who analyse the data. This refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

5. VERACITY

The quality of the data being captured can vary greatly. Accuracy of analysis depends on the veracity of the source data.

6. COMPLEXITY

Data management can become a very complex process, especially when large volumes of data come from multiple sources. These data need to be linked, connected and correlated in order to be able to grasp the information that is supposed to be conveyed by these data. This situation, is therefore, termed as the 'complexity' of Big Data.

SO IS HADOOP A SOLUTION?

Hadoop is a framework of software libraries designed in order to handle this problem of Big Data. It is capable of efficiently processing large volumes of Unstructured Data. It can grow limitlessly and hence provides a highly scalable distributed environment as a solution to all our data problems.

We will talk about Hadoop in more details in the next chapter.

INTRODUCTION TO HADOOP

With Big Data in picture it was soon realised that the Traditional Distributed Systems are not capable enough of handling the increasing demand of Data Analysis and Maintenance.

Google being the largest source of data was the first one to feel the pinch of this situation and came up with a new version of distributed architecture called Google File System or GFS. Google realised that it was not the only one that would be suffering from the Big Data problem so it generously released the White Papers of the GFS in to the Open Source Community.

White Paper or Research Paper is a document which defines a problem, defects in the existing solutions, new proposed solution and some architectural features of the proposed system. An open source community is a group of contributors that develop technological stuff for free and make it available to the general masses free of cost.

A Yahoo developer called Doug Cutting along with the largest open source community the Apache build a new framework called as HADOOP. This framework was based on the working and design principles laid down by GFS.

WHAT IS HADOOP?

Apache Hadoop is a framework of software libraries designed on top of a distributed platform to help solve the problem of Big Data Analysis. In simple words Apache Hadoop is a framework of software libraries written in Java, designed with a programming model and storage structure to handle large data sets efficiently in a distributed architecture.

A framework is a set of programs, instructions, rules and regulations designed in some programming language, bundled as the library. Every framework has a defined platform and a defined way of working in its defined architecture.

HADOOP ARCHITECTURE

In this section we will understand the different components of a Hadoop Cluster. We will try to logically setup the cluster and then learn the different features of it. We will also see how the cluster is broken up into two core components viz. HDFS and Map Reduce.

GETTING THE SERVERS

For our Hadoop Architecture we will buy 4 Servers from the market. They can be physical or cloud servers *(Mostly Cloud Servers are used in Production Deployments)*.

A Server is a high end computer which is used for installation of dedicated frameworks or applications. A server can be configured as Master or Slave. A server has Storage, RAM, and CPU. It does not have Input Output peripherals.

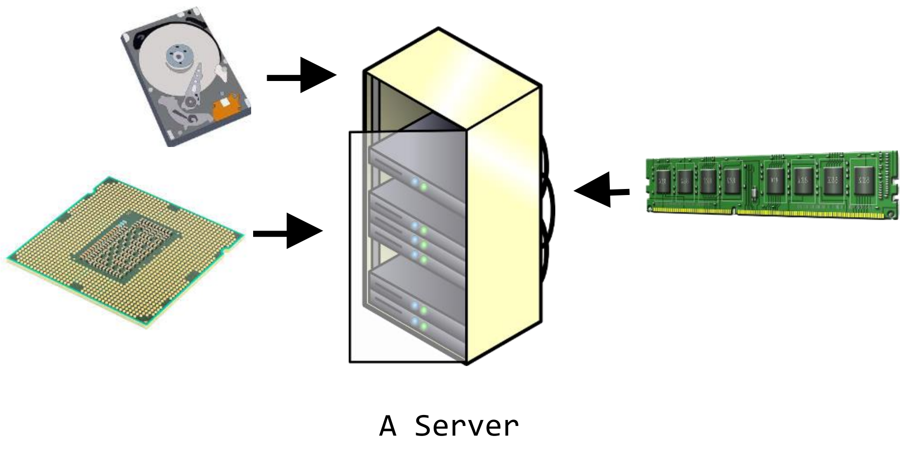
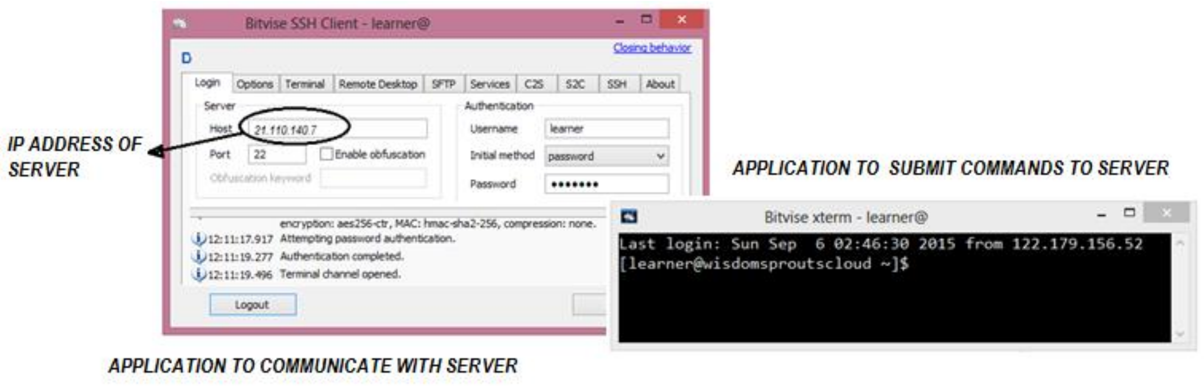


Fig 6: Server

COMMUNICATING WITH THE SERVERS

Using our Laptop we will establish communication with the servers using IP address of each of them and build a network among them. For this purpose we can use any communication client like Putty. Here after the laptop will become our client.


IP ADDRESS OF SERVER
APPLICATION TO SUBMIT COMMANDS TO SERVER
APPLICATION TO COMMUNICATE WITH SERVER
Bitvise SSH Client - learner@
Closing behavior
Login | Options | Terminal | Remote Desktop | SFTP | Services | C2S | S2C | SSH | About
Logout
Bitvise xterm - learner@

Server
Host 21.110.140.7
Port 22 ☐ Enable obfuscation
Obfuscation keyword

Authentication
Username learner
Initial method password
Password •••••••

```
encryption: aes256-ctr, MAC: hmac-sha2-256, compression: none.
12:11:17.917 Attempting password authentication.
12:11:19.277 Authentication completed.
12:11:19.496 Terminal channel opened.
```

```
Last login: Sun Sep  6 02:46:30 2015 from 122.179.156.52
[learner@wisdomsproutscloud ~]$
```

Fig 2: Client Application for communication with Server

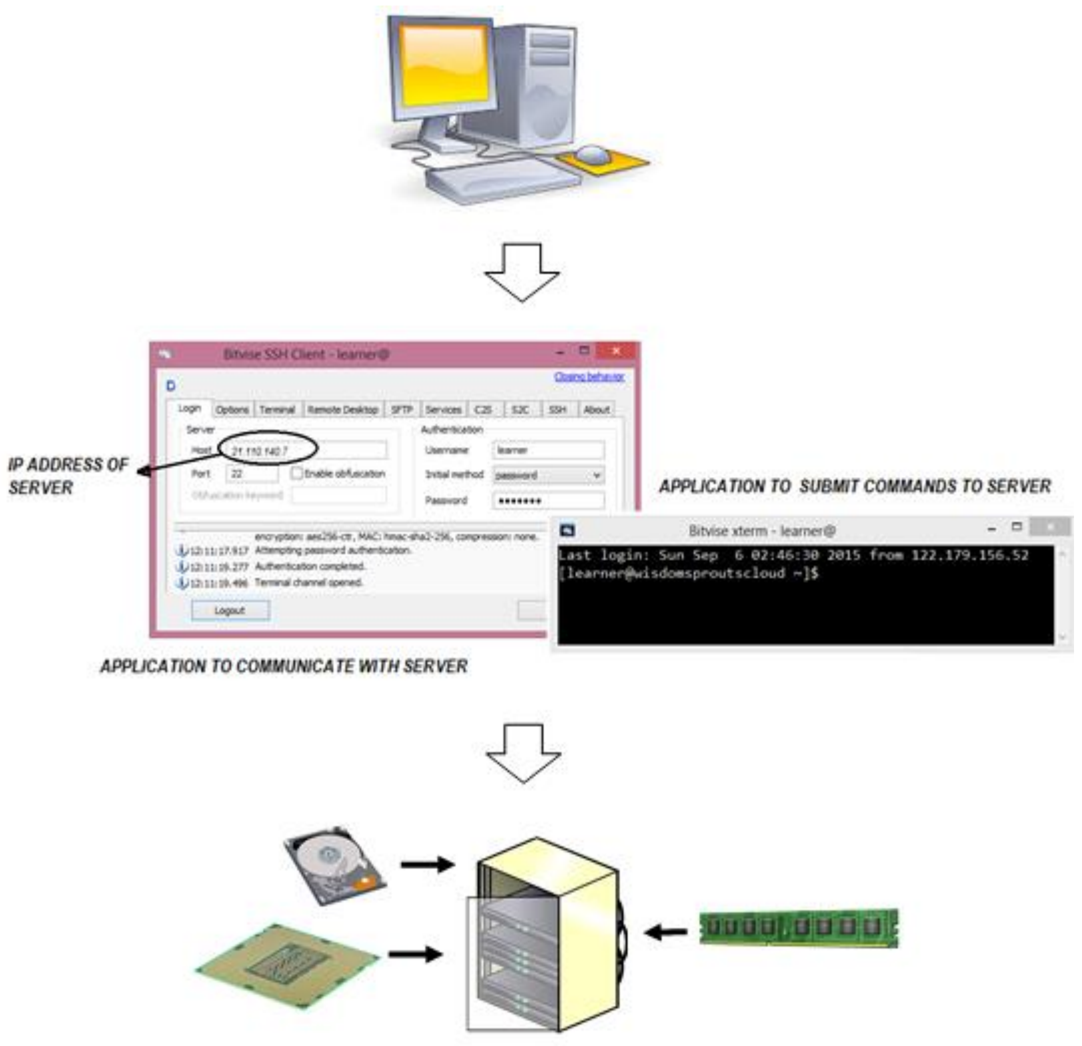

IP ADDRESS OF SERVER
APPLICATION TO SUBMIT COMMANDS TO SERVER
APPLICATION TO COMMUNICATE WITH SERVER
Bitvise SSH Client - learner@
Bitvise xterm - learner@

Fig 7: Client Server Communication

INSTALLING OPERATING SYSTEM

Now, we will install an operation system in each of the servers through the client. Hadoop can be configured on Linux as well as Windows platform. In this chapter we would be using the Linux compatible version of Hadoop.

Any version of Linux like RHEL, Centos, Fedora or Ubuntu can be used. In the production deployments, RHEL and Centos are most commonly preferred. As a part of ideal configurations of the dedicated servers it is recommended to reserve 30% of the total storage space of the servers for the Operating System.

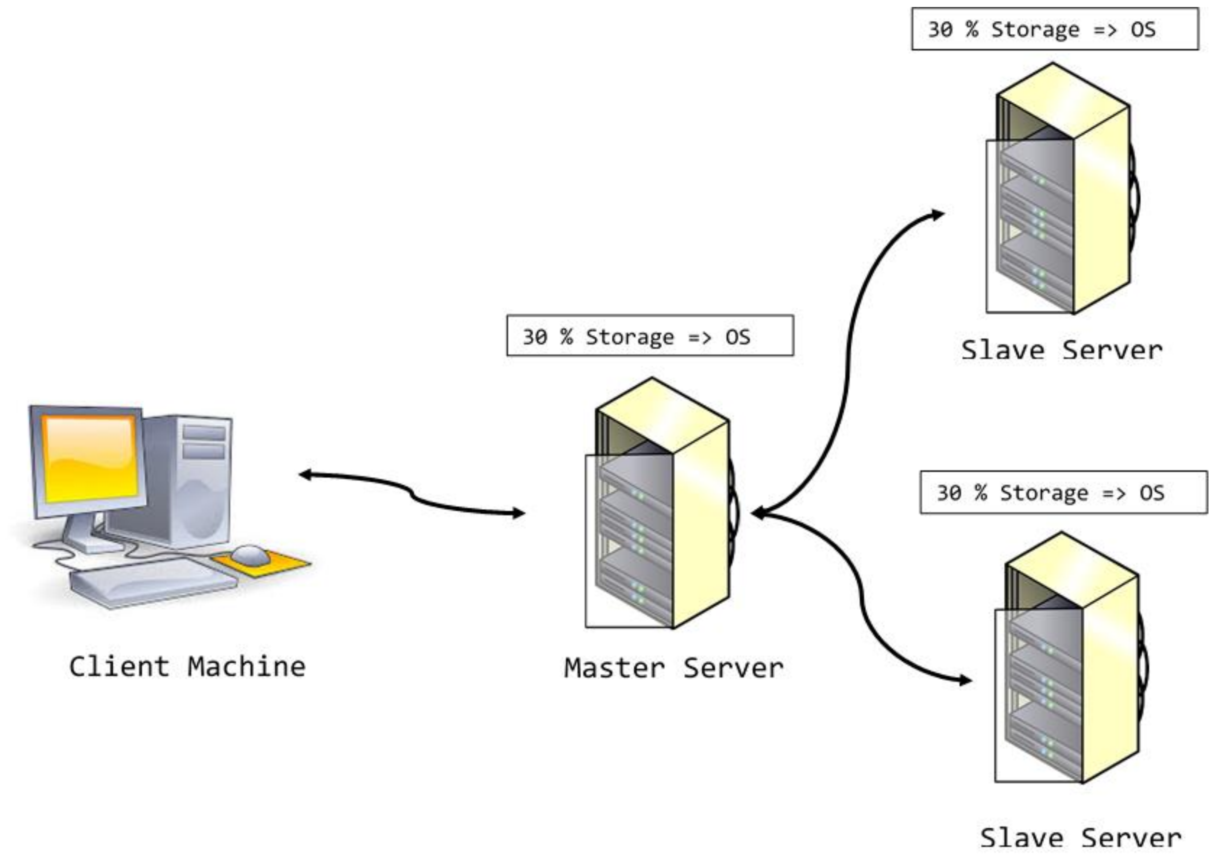


Fig 8: OS Installation

HADOOP INSTALLATION

Basically there are two available variants of Hadoop viz. Core or open source Hadoop *(Apache)* and Enterprise Hadoop platform *(Cloudera and Hortonworks)*. For Installing Hadoop we will be downloading the Hadoop Packages from the official Apache or Cloudera website. The download and installation commands and procedure will depend on the type of Operating system used. In this chapter we would be dealing with Core Hadoop platform.

Installation of Core Hadoop gives you two things

1. HDFS or Hadoop Distributed File System *(Storage Structure)*
2. Map Reduce *(Processing Model)*

HADOOP NODE

A server where the two core components of Hadoop viz. HDFS and Map Reduce are available or where Hadoop is installed is called as a Hadoop Node.

After the installation we will get the following:

1. Hadoop Jar files
2. Hadoop Xml files
3. Hadoop Daemons

HADOOP JAR FILES

Hadoop being based on Java, all the instructions and default programs are bundled into these jar files. These jar files will guide the Hadoop Daemons on how to perform any operation or function. All the default input/output formats, programming phases, base classes are covered under these files.

HADOOP XML FILES

For the ease of changing certain configurations and settings these xml files are being used. Basically there are 3 xml files in a Hadoop installation

1. CORE-SITE.XML

This file is used for making any changes related to the overall Hadoop Cluster. Mostly network and hardware settings.

2. HDFS-SITE.XML

This file is used for making any changes to the distributed file system or storage of Hadoop. For example: changing the default block size or modifying the replication factor.

3. MAPRED-SITE.XML

This file is used for changing configuration related to the programming model of Hadoop i.e. Map Reduce. For example: changing the default number of Reducers or Mappers.

HADOOP DAEMONS

Hadoop installation provides daemons or services or processes that use the “Java Virtual Machine” (JVM) for their functioning.

The Java virtual machines work as a container for the services to perform their actions.

There are total 5 services or daemons that take care of all Hadoop operations like storage/retrieval of data and processing that data.

1. NAME NODE

This service acts like a master driver for the HDFS. This will manage all the operations related to storage. This service is installed on the strongest server in the cluster. Name Node is considered to be the Brain of the system. Name Node going down will leave the entire cluster inaccessible. The major responsibilities of the Name Node include

1. Managing client request to store/retrieve data/results.
2. Managing data distribution among all the Data Nodes.
3. Managing failovers of Data Nodes

2. DATA NODE

This daemon acts like a slave driver for the HDFS. The Data Node is the one actually responsible for carrying out the storage related operations under the instructed guidance of the Name Node. Major responsibilities of Data Node include

1. Managing 70% storage space in the server as HDFS.
2. Storing data from client in HDFS.

3. JOB TRACKER

This daemon acts like a master driver for Map Reduce. This service will manage all the Map Reduce related operations. This service can be installed on the same server as the Name Node or potentially on a separate master server. If the Job Tracker goes down, all the running and queued jobs need to be resubmitted. The major responsibilities of Job Tracker are

1. Splitting the client job into small tasks.
2. Scheduling and managing the tasks execution on the Task Trackers
3. Failure handling on the Task Trackers.

4. TASK TRACKER

This daemon acts like a slave driver for Map Reduce. This service is responsible for executing Map Reduce tasks. Task Tracker and Data Nodes are part of every slave node and will always be local to each other. Task Tracker will operate only on the data that is stored by the Data Node in the same server where the Task Tracker is.

5. SECONDARY NAME NODE

This daemon is one of the master services of HDFS. It acts like a cold backup for the Name Node. This service will store the Mappings created by the Name Node. In case the Name Node fails, the Secondary Name Node **will not** become the Name Node. It will help you restore data in the new Name Node or in the same Name Node when it recovers from the failure.

It's moreover like a cloud storage backup for your Hard Disk Drive. If your HDD fails, then you have to buy a new one and then you can take the backup from the cloud storage. It is not a failover and is there only to ensure that the Name Node's data is replicated somewhere.

CONFIGURING THE MASTER AND SLAVE NODES

We will install the Hadoop Master package on the server that we want to use as a Master. This will give us the Name Node and Job Tracker services on the Master. Then we will install the hadoop Slave package on rest of the Servers. This will give us the Data Node and Task Tracker services on each of the slave nodes.

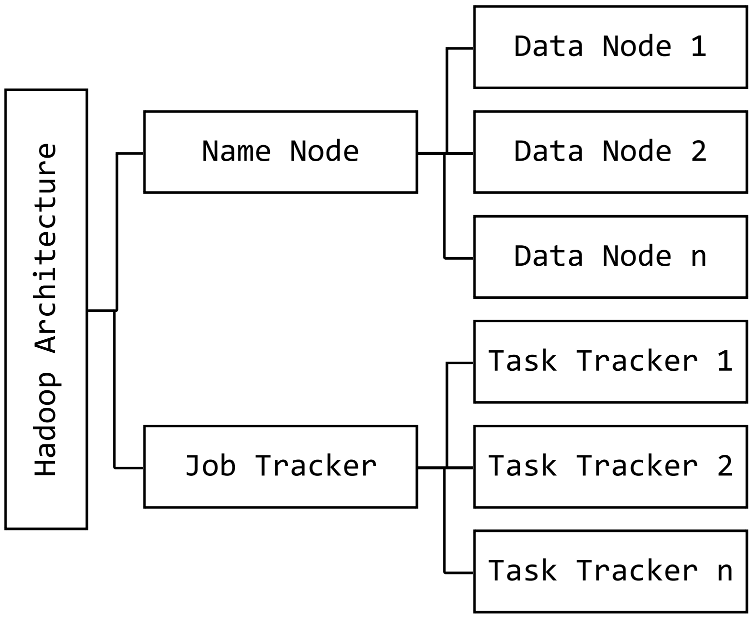


Fig 9: Hadoop Architecture

CONFIGURING HADOOP CLIENT

We will install Hadoop Client package on our laptop so that we can use it to directly communicate with our Hadoop cluster. This client will be used to send commands to the Hadoop Cluster, load and retrieve data from HDFS, monitor and maintain health and status of the cluster. It will contain all the basic configuration files of the Hadoop Cluster.

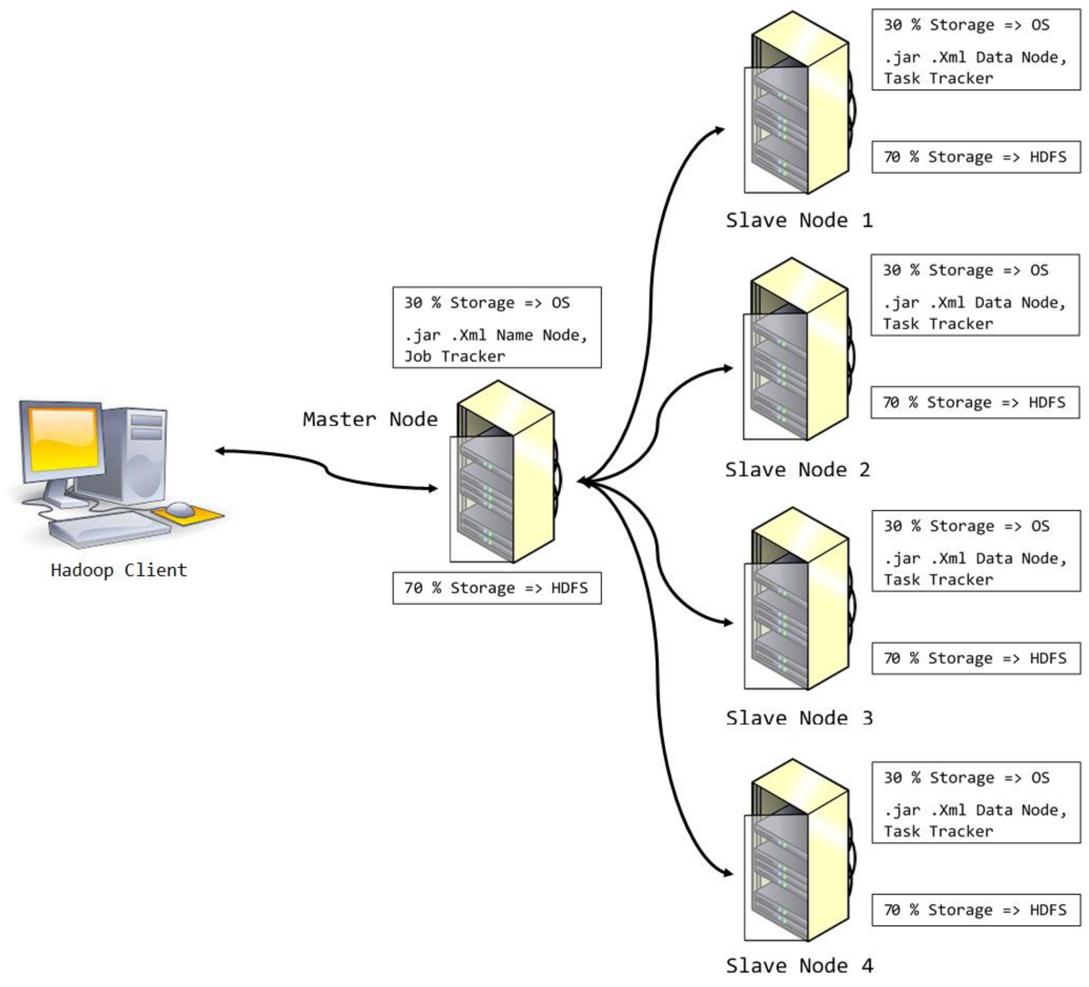


Fig 10: Hadoop Cluster

HADOOP CLUSTER

A group of Hadoop Nodes is called as a Hadoop Cluster. A cluster can have Nodes ranging from 1 to 4,000 (Maximum number of nodes based on performance testing). Hadoop can be installed and configured in two modes

1. PSEUDO DISTRIBUTED MODE

In this mode all the master and slave services along with the client reside in a single system. This mode is useful for testing and learning purpose but won't take you too far as you are dependent on the scaling capabilities of a single machine.

2. DISTRIBUTED MODE

This is the actual mode for which Hadoop was built. In this mode the master services reside on separate servers and slave services reside on separate servers. The most efficient part of this mode is that commodity hardware can be used for configuring the slave nodes. In this mode we can dynamically scale horizontally.

Commodity hardware means cheap hardware and career class hardware means expensive hardware.

HADOOP DISTRIBUTED FILE SYSTEM

HDFS or Hadoop Distributed File System is the storage unit available in Hadoop. It is the combined storage of the 70% HDFS from each slave node.

For example: If the storage capacity of each slave node is 10 TB then every slave node will have HDFS capacity of 7 TB. In this scenario the HDFS capacity of the Hadoop Cluster will be 70 TB, provided we have 10 slave nodes (HDFS of Name Node is not used for storing data).

HDFS is a read only file system. Once data is inserted into HDFS, it cannot be modified. All data in HDFS is stored in text format in a flat file system. Images and Videos are converted in matrix of pixel which is again stored as text file. It is more or less like our computers where everything is stored in Binary format.

It is managed by the Name Node and Data Node services where Name Node decides which block of data is to be stored on which slave node and then the corresponding Data Nodes stores the data in the HDFS of that node.

MAP REDUCE

Map Reduce is the basic and only programming model available in Hadoop. The Job Tracker and Task Tracker services handle this model. The Job Tracker schedules and monitor the component tasks in a Map Reduce Job on the Task Trackers where the actual programming happens on the local data available with the corresponding Data Nodes in the same server. The programs are bundled as Jar files consisting of three Java Class files

1. MAP CLASS

It consists the Map Function. User can define the programming logic for this function.

2. REDUCE CLASS

It consists the Reduce Function. User can define the programming logic for this function.

3. MAIN CLASS

It defines the Map and the Reduce classes. User can define the programming logic for this function. This is where all the exception handling is done and all the configurations are specified.

In the next chapter we will have a look at the different aspects in the Job Process of Hadoop File Storage.

HADOOP JOB PROCESS – HDFS

In the last chapter we logically saw how to setup a Hadoop Cluster. In this chapter let's try and put some files into it.

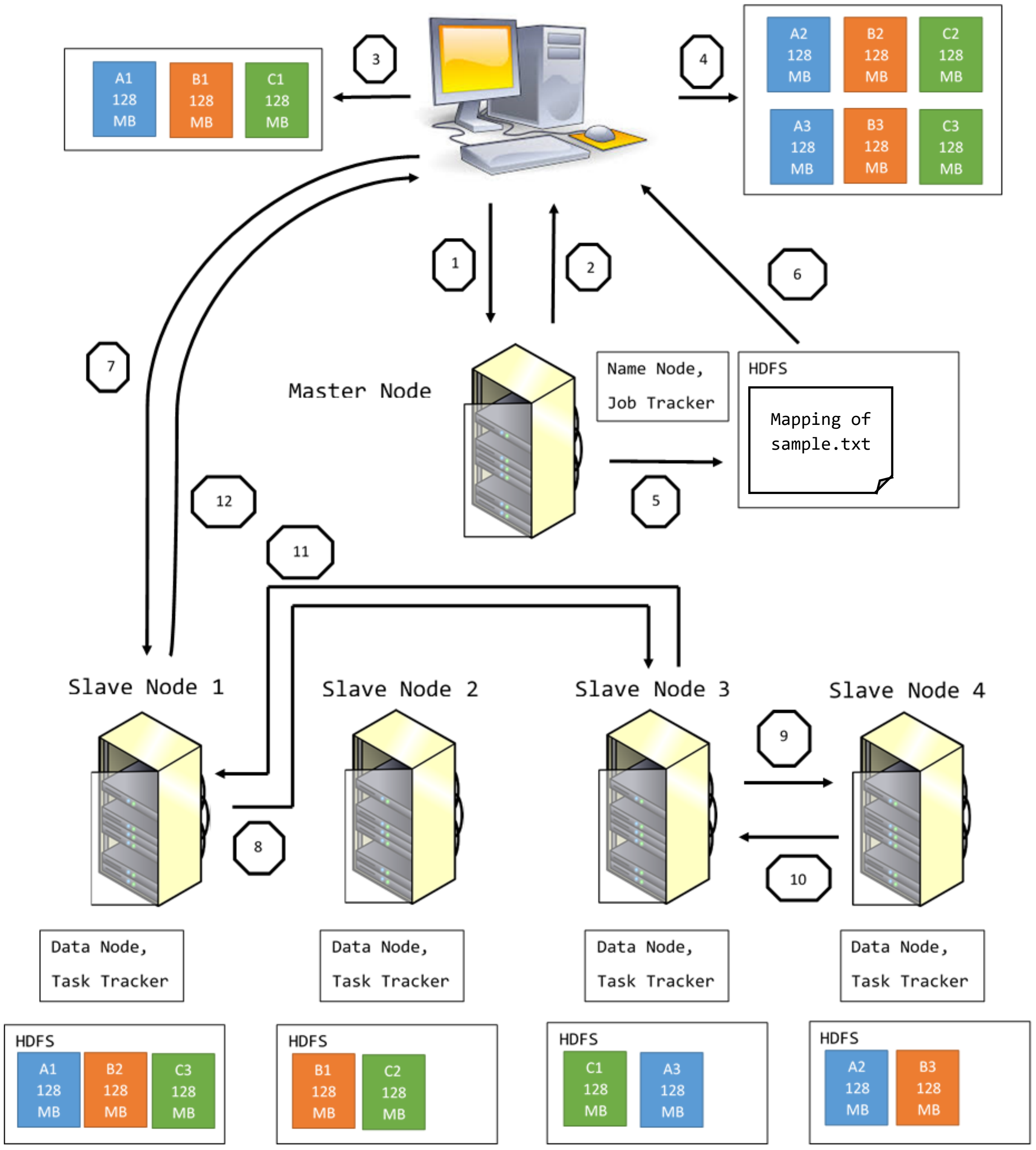


Fig 11: File Anatomy of Write

STEP 1 AND 2: CLIENT SIDE INFORMATION

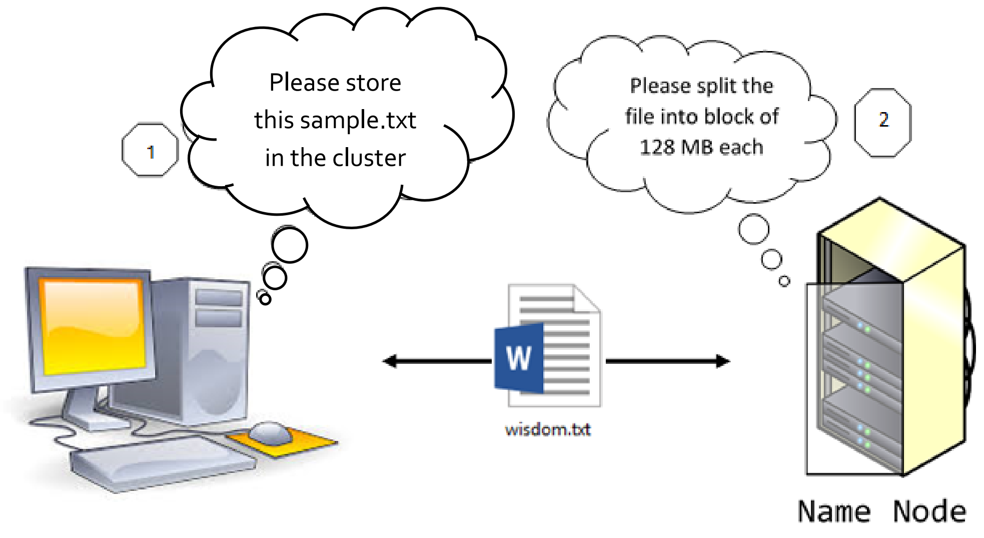
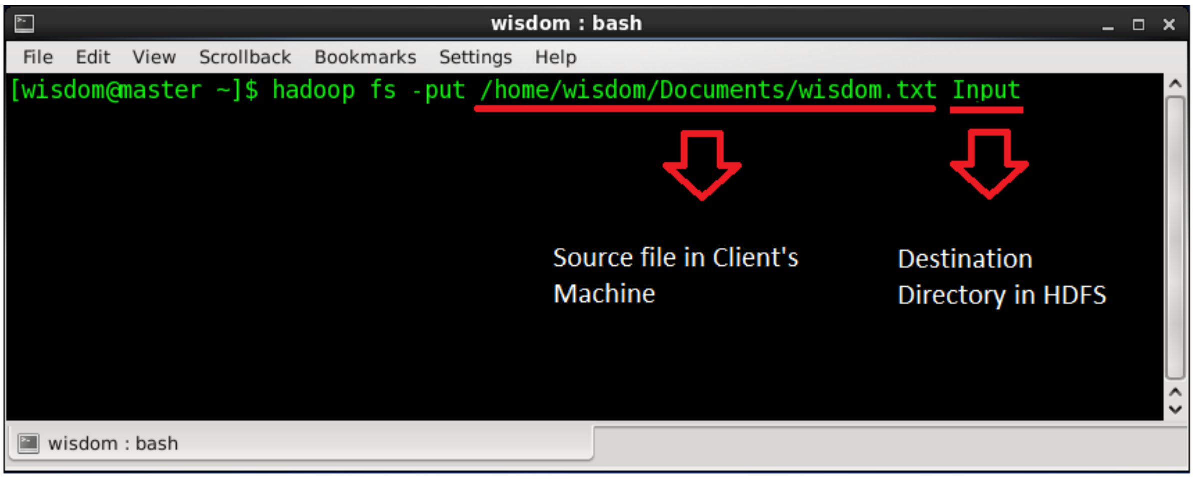

Please store this sample.txt in the cluster
1
Please split the file into block of 128 MB each
2
wisdom.txt
Name Node

Fig 12: File Storage Request

The client has a file named sample.txt of size 300 MB which is to be stored on the cluster. The client will issue a command for the same which will be responded by the master of storage viz. Name Node. The Name Node will then consult with the Jar files and figure out the steps for the file storage.


wisdom : bash
File Edit View Scrollback Bookmarks Settings Help
Source file in Client's Machine
Destination Directory in HDFS
wisdom : bash

```
[wisdom@master ~]$ hadoop fs -put /home/wisdom/Documents/wisdom.txt Input
```

Fig 13: File Storage Command

STEP 3: FILE SPLIT

Based on this the Name Node will instruct the client to split the file into blocks of 128 MB each. So we will have 3 blocks A1, B1 and C1 with size 128 MB, 128 MB and 44 MB respectively.

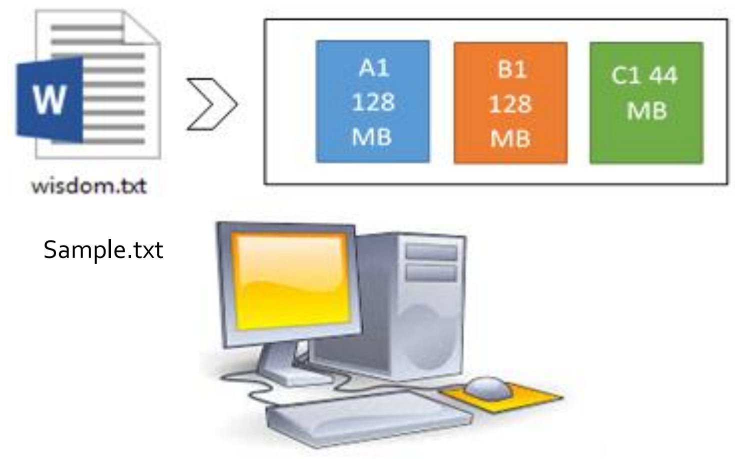


Fig 14: File Split into

WHAT EXACTLY ARE HADOOP BLOCKS?

When you store a file in HDFS, the system breaks it down into a set of individual blocks and stores these blocks in various slave nodes in the Hadoop cluster. This helps in achieving distributed scaling. The concept of storing a file as a collection of blocks is entirely consistent with how file systems normally work. But what’s different about HDFS is the scale. A typical block size that you’d see in a file system under Linux is 4KB, whereas a typical block size in Hadoop is 128MB. This value is configurable, and it can be customized, as both a new system default and a custom value for individual files.

ADVANTAGES OF HADOOP BLOCKS

1. The blocks are of fixed size, so it is very easy to calculate the number of blocks that can be stored on a disk.
2. HDFS block concept simplifies the storage of the Data Nodes. The Data Nodes doesn’t need to concern about the blocks metadata data like file permissions etc. The name Node maintains the metadata of all the blocks.

3. If the size of the file is less than the HDFS block size, then the entire file is treated as one block.
4. As the file is chunked into blocks, it is easy to store a file that is larger than the disk size as the data blocks are distributed and stored on multiple nodes in a hadoop cluster.
5. Blocks are easy to replicate between the Data Nodes and thus provide fault tolerance and high availability.

STEP 4: BLOCK REPLICATION

Further the Name Node will instruct the client to replicate each block based on the value of the Replication Factor. So now, we will have

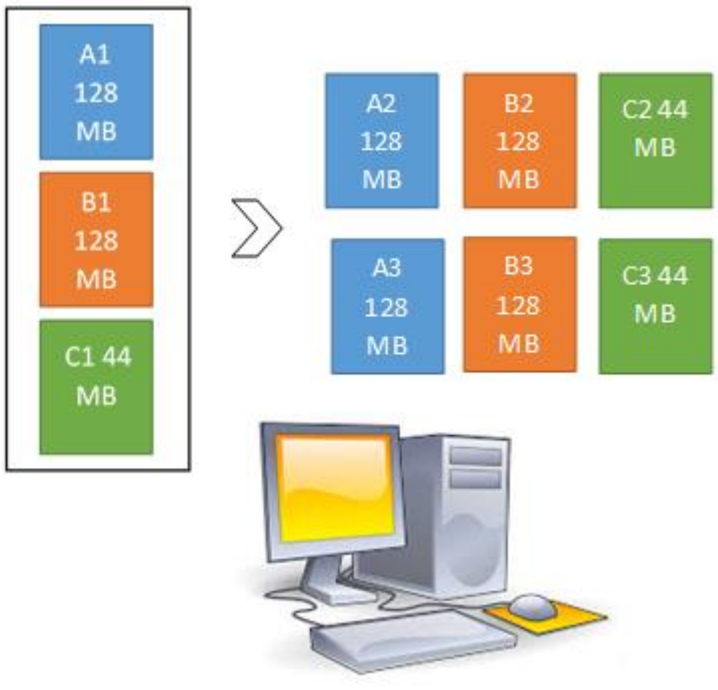


Fig 15: Replicated Blocks

WHAT EXACTLY IS HADOOP REPLICATION?

In Hadoop the data blocks are distributed amongst all Data Nodes. To preserve the data block after failure, every block is duplicate in some numbers based on a property of HDFS blocks called as Replication factor.

The default value of Replication Factor is 3, which means that there should be a total number of 3 copies of each block which will be distributed based on Name Node’s mapping. The value of replication factor can be lowered or increased base on how critical your data is or based on how failure prone your servers are.

If a data block is lost due to the failure of the node in which it was stored, the replica from some other Node can be utilized and we don't have anything to worry about.

STEP 5: BLOCK MAPPING

Now the Name Node will create mappings for the sample.txt file and instruct the Client to distribute the blocks among the Data Nodes in reference to the mapping.

Data Node 1	Data Node 2	Data Node 3	Data Node 4
Block A1	Block B1	Block C1	Block A2
Block B2	Block C2	Block A3	Block B3
Block C3			

Fig 16: Name Node Mapping

WHAT EXACTLY IS THE NAME NODE MAPPING?

After a file is split into blocks and after the blocks are replicated, the Name Node creates a mapping of the blocks and the available Data Nodes. This Mapping defines which block will be stored on which slave node.

Name Node randomly distributes the blocks among its Data Nodes with only one consideration that, in an ideal scenario, no two copies of the same block are stored on the same server as this is not leave any meaning to replication because if one server goes down then all the copies of the block that was stored on the server are lost.

The Name Node stores this mapping in its HDFS and provides the same to the client for data retrieval and to the Job Tracker for task distribution.

If this mappings are lost, the data will still be available in the Data Nodes but nobody will be aware of which block is in which Data Node and which blocks together form a requested file. So the entire cluster will go down.

Hence the Name Node is called as the Brain of the Cluster. A copy of this Mapping is stored in the Secondary Name Node.

STEP 6 AND 7: BLOCK DISTRIBUTION - CLIENT

The Name Node then submits a temporary copy of this Mapping to the client and instructs it to distribute the blocks accordingly.

The Client then establishes connection with the nearest Data Node in the network and distributes a complete block along with the replicas and the respective mapping to it and instructs it to do the distribution accordingly.

For Example: The client will connect to Data Node 1 and distribute all copies of A viz. A1, A2, A3 to it and instruct it to keep one copy with itself and distribute one to Data Node 3 and one to Data Node 4.

STEP 8 AND 9: BLOCK DISTRIBUTION – DATA NODE

Data Node will store Block A1 into its HDFS. Data Node 1 will then establish connection with the nearest Data Node out of Data Nodes 3 and 4 (let's assume Data Node 3) and distribute two copies of A (A2 and A3)to it.

Data Node 1 will instruct Data Node 3 to keep one copy of A with itself and distribute the remaining one to Data Node 4.

Data Node 3 as instructed will keep one copy of A (A2) in its HDFS and distribute the other (A3) to Data Node 4.

STEP 10, 11 AND 12: BLOCK DISTRIBUTION - ACKNOWLEDGEMENT

Data Node 4 after storing the block (A3) in its HDFS will acknowledge Data Node 3 about the same who will acknowledge Data Node 1, who in turn will acknowledge the Client about the success of the block storage for block A and its replicas.

This procedure will repeat for all the block after completion of which the client will intimate the Name Node about the same.

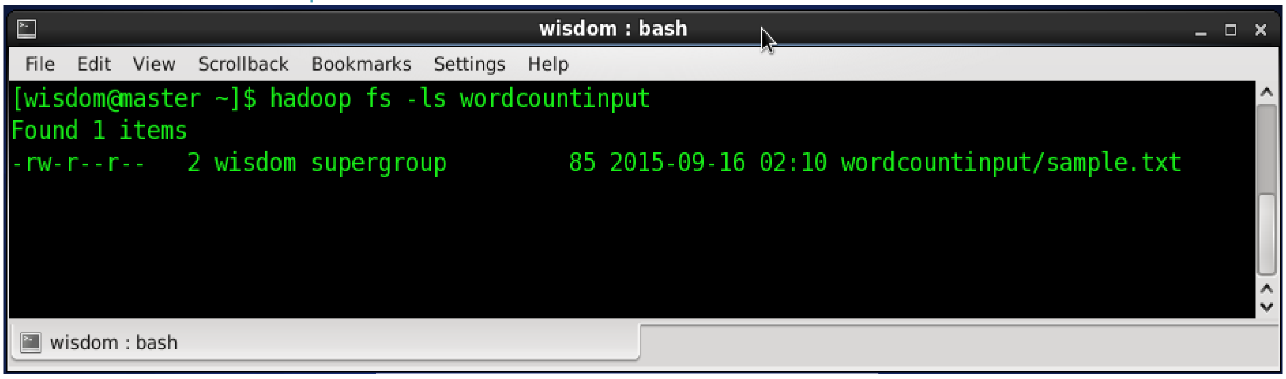
This process is also called as File Anatomy of Write. File Anatomy of Read is the exact opposite of this process.

In the next chapter we will have a look at the different aspects in the Job Process of Hadoop File Processing.

HADOOP JOB PROCESS – MAP REDUCE

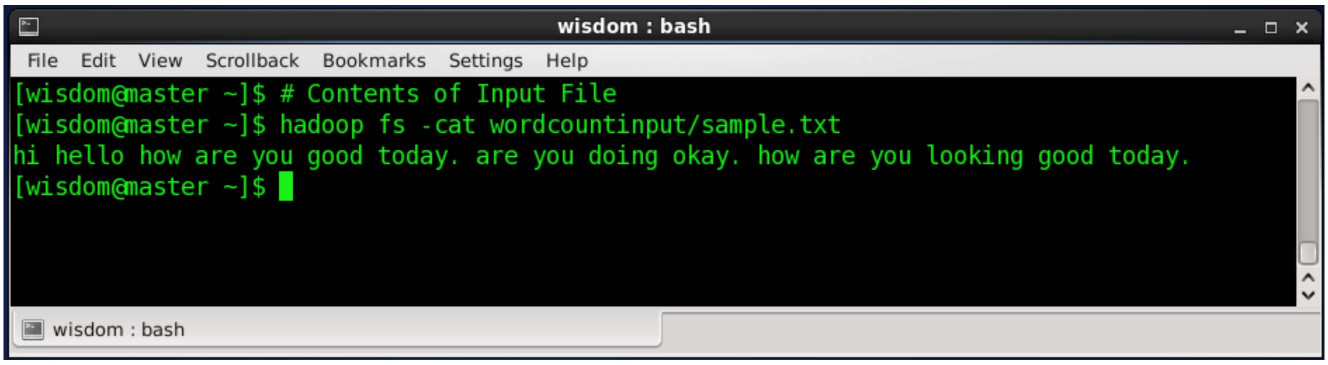
In the last chapter we saw details about how a file is stored in the Hadoop Cluster. In this chapter let's see how we can run a sample Map Reduce program on a file stored in the Hadoop Cluster.

Let's assume our client wants to execute a Word Count program bundled as wordcount.jar on a file called as sample.txt stored in a directory on HDFS called wordcountinput.



```
[wisdom@master ~]$ hadoop fs -ls wordcountinput
Found 1 items
-rw-r--r--   2 wisdom supergroup         85 2015-09-16 02:10 wordcountinput/sample.txt
```

Fig 17: Input Folder



```
[wisdom@master ~]$ # Contents of Input File
[wisdom@master ~]$ hadoop fs -cat wordcountinput/sample.txt
hi hello how are you good today. are you doing okay. how are you looking good today.
[wisdom@master ~]$
```

Fig 18: Input File

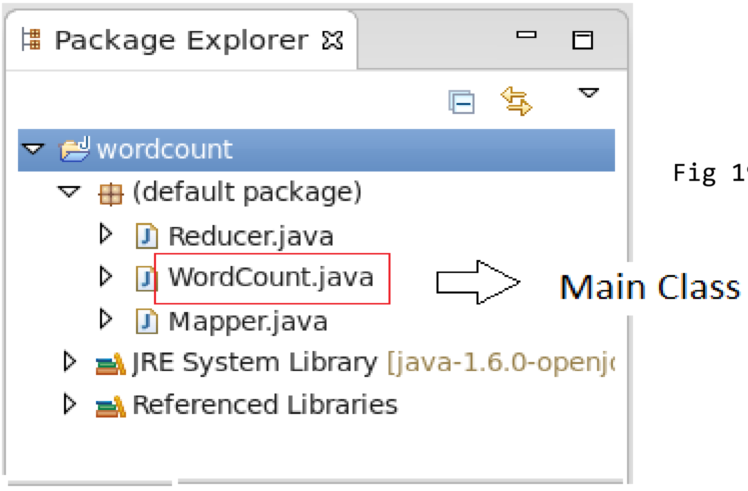
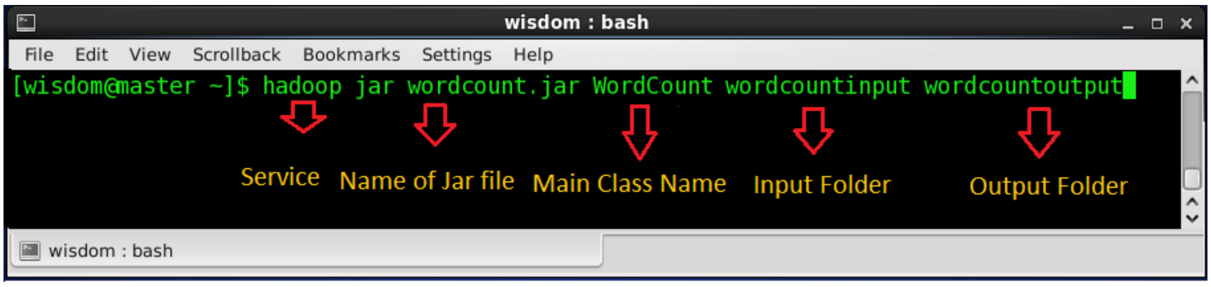


Fig 19: Jar File

PROGRAM SUBMISSION

The client submits the program called as wordcount.jar that is to be executed on the file sample.txt.



```
[wisdom@master ~]$ hadoop jar wordcount.jar WordCount wordcountinput wordcountoutput
```

Fig 20: Job Submission

JOB TRACKER'S ROLE

The Job Tracker being the master of Map Reduce will respond to this command and will consult the Name Node for details of the file sample.txt.

MAPPING

From the Name Node's mapping (Illustrated in the last chapter) the Job Tracker will realize that if it has to perform any operation on file sample.txt means it has to perform that operation on any one copy of A, any one copy of B and any one copy of C.

Data Node 1	Data Node 2	Data Node 3	Data Node 4
Block A1	Block B1	Block C1	Block A2
Block B2	Block C2	Block A3	Block B3
Block C3			

Fig 21: Mapping of sample.txt

JOB SPLIT IN TO TASK

The Job Tracker will consult the Installation Jar files to verify the configurations related to the program execution specified in them. Then it will check if the client has submitted any configurations in the command. If yes then this settings will override the setting provided by the Installation Jar files. Then the client will check for configurations in the Main Class file of the program. The settings specified in the program will override all other settings.

With this the Job Tracker will realize the number of Map and Reduce phases to be executed and will split the Job as follow

1. MAP PHASE

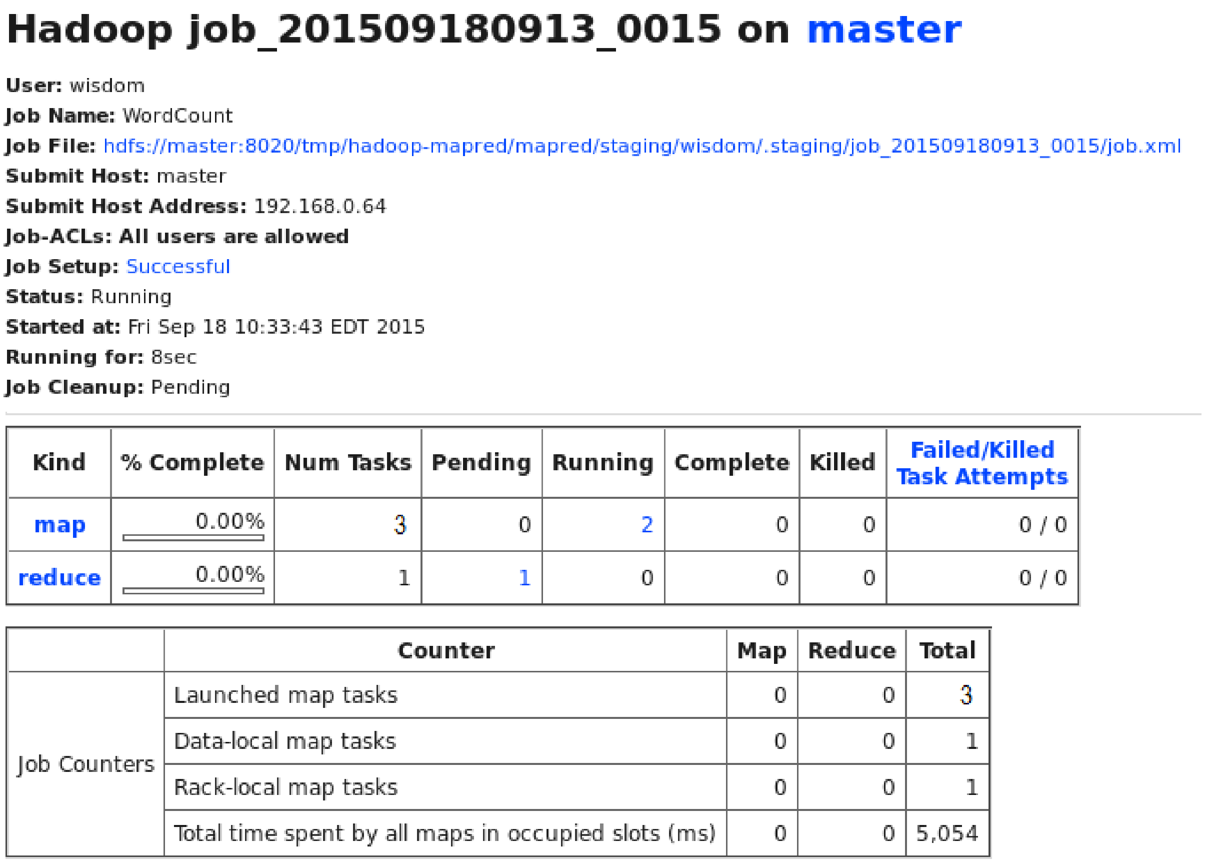
Task 1: Execute Map (any copy of A)

Task 2: Execute Map (any copy of B)

Task 3: Execute Map (any copy of C)

2. REDUCE PHASE

Task 4: Execute Map ([Output of Task 1] + [Output of Task 2] + [Output of Task 3])



Hadoop job_201509180913_0015 on master

User: wisdom
Job Name: WordCount
Job File: hdfs://master:8020/tmp/hadoop-mapred/mapred/staging/wisdom/.staging/job_201509180913_0015/job.xml
Submit Host: master
Submit Host Address: 192.168.0.64
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Running
Started at: Fri Sep 18 10:33:43 EDT 2015
Running for: 8sec
Job Cleanup: Pending

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	0.00%	3	0	2	0	0	0 / 0
reduce	0.00%	1	1	0	0	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	Launched map tasks	0	0	3
	Data-local map tasks	0	0	1
	Rack-local map tasks	0	0	1
	Total time spent by all maps in occupied slots (ms)	0	0	5,054

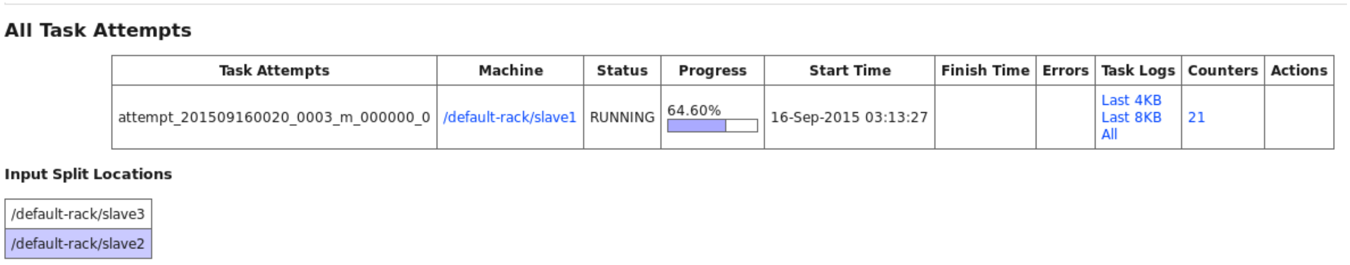
Fig 22: Job Split into Tasks

JOB SCHEDULING

The Job Tracker will then analyse the number of execution slots required for each task.

Execution slots is a combination of some proportion of resources like Storage, RAM and CPU.

Based on the availability of the required data block and the required number of execution slots, the Job Tracker will randomly allocate all tasks of Map Phase to the available Task Trackers.


All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Finish Time	Errors	Task Logs	Counters	Actions
attempt_201509160020_0003_m_000000_0	/default-rack/slave1	RUNNING	64.60%	16-Sep-2015 03:13:27			Last 4KB Last 8KB All	21	

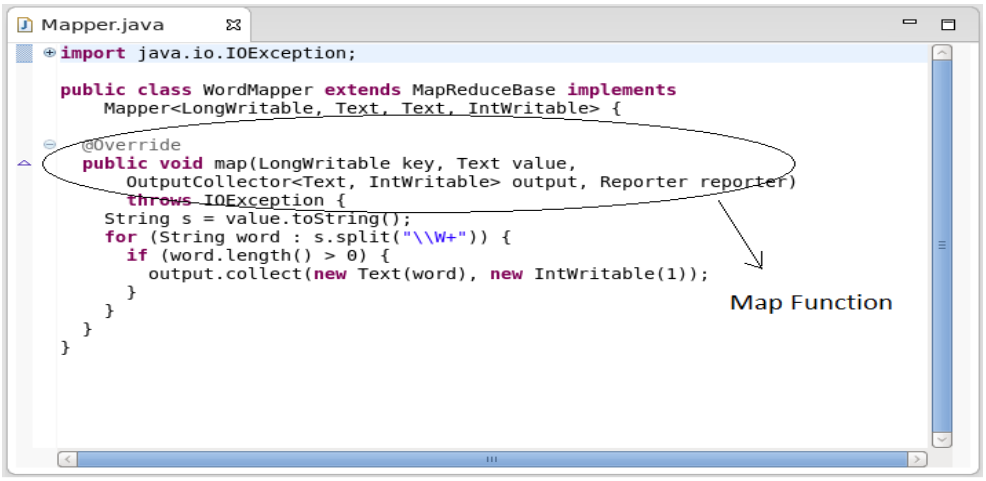
Input Split Locations

/default-rack/slave3
/default-rack/slave2

Fig 23: Task Allotment

MAPPER IN ACTION

The time when a Task Tracker is in the Map Phase or when it is executing a Map function, it is called as the Mapper. Every Mapper produces one output file by executing the Map function on the local data block in its HDFS. The output file is an intermediate result and is stored in the temporary folders.

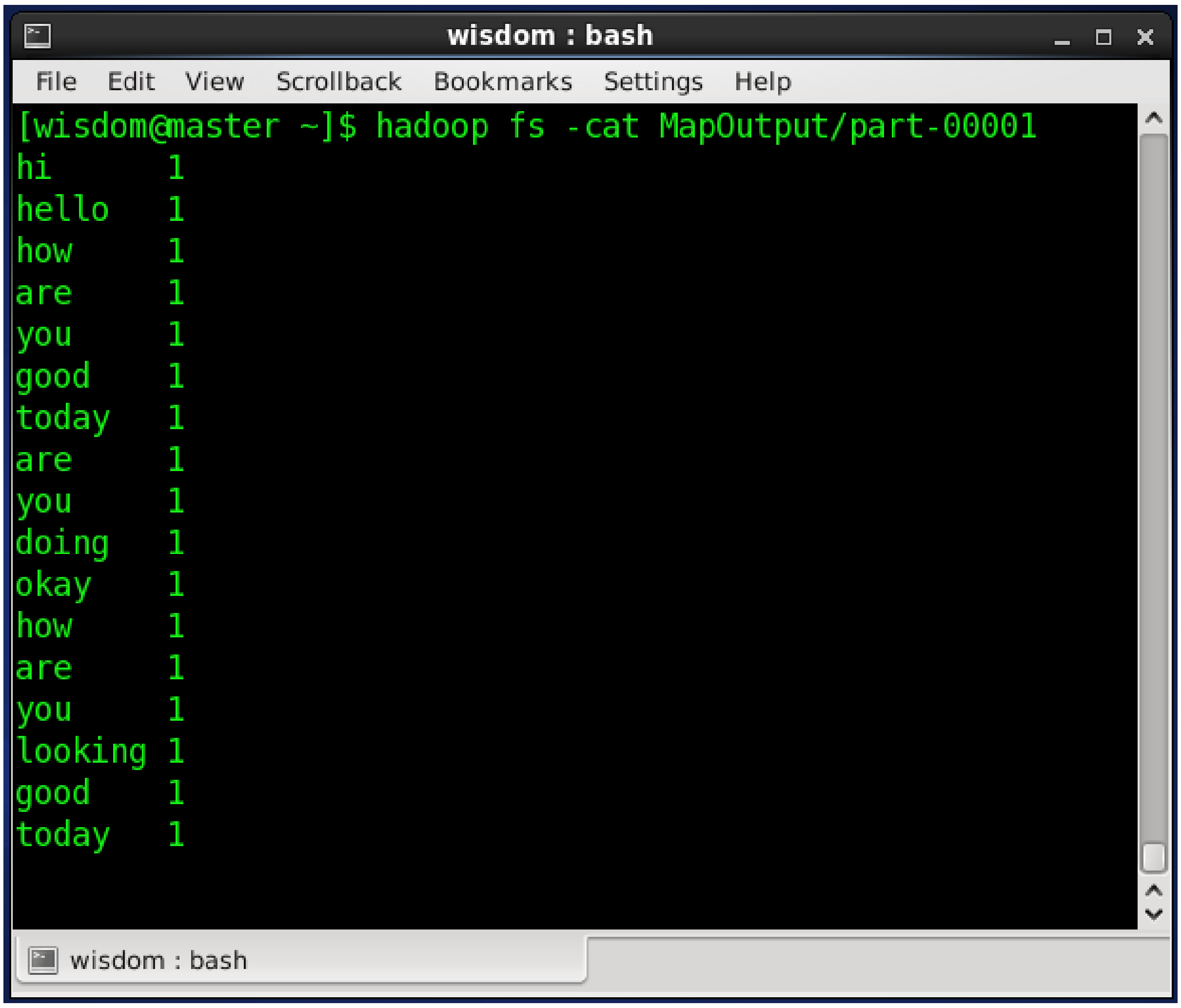


```
import java.io.IOException;

public class WordMapper extends MapReduceBase implements
    Mapper<LongWritable, Text, Text, IntWritable> {

  @Override
  public void map(LongWritable key, Text value,
      OutputCollector<Text, IntWritable> output, Reporter reporter)
      throws IOException {
    String s = value.toString();
    for (String word : s.split("\\W+")) {
      if (word.length() > 0) {
        output.collect(new Text(word), new IntWritable(1));
      }
    }
  }
}
```

Fig 24: Map Function

Generally in the Map phase, different keys in the input are associated with some values. This is called as mapping.

In our case, the words are keys and the value that we are mapping them to is '1'.



```
[wisdom@master ~]$ hadoop fs -cat MapOutput/part-00001
hi      1
hello   1
how     1
are     1
you     1
good    1
today   1
are     1
you     1
doing   1
okay    1
how     1
are     1
you     1
looking 1
good    1
today   1
```

Fig 25: Mapper Output

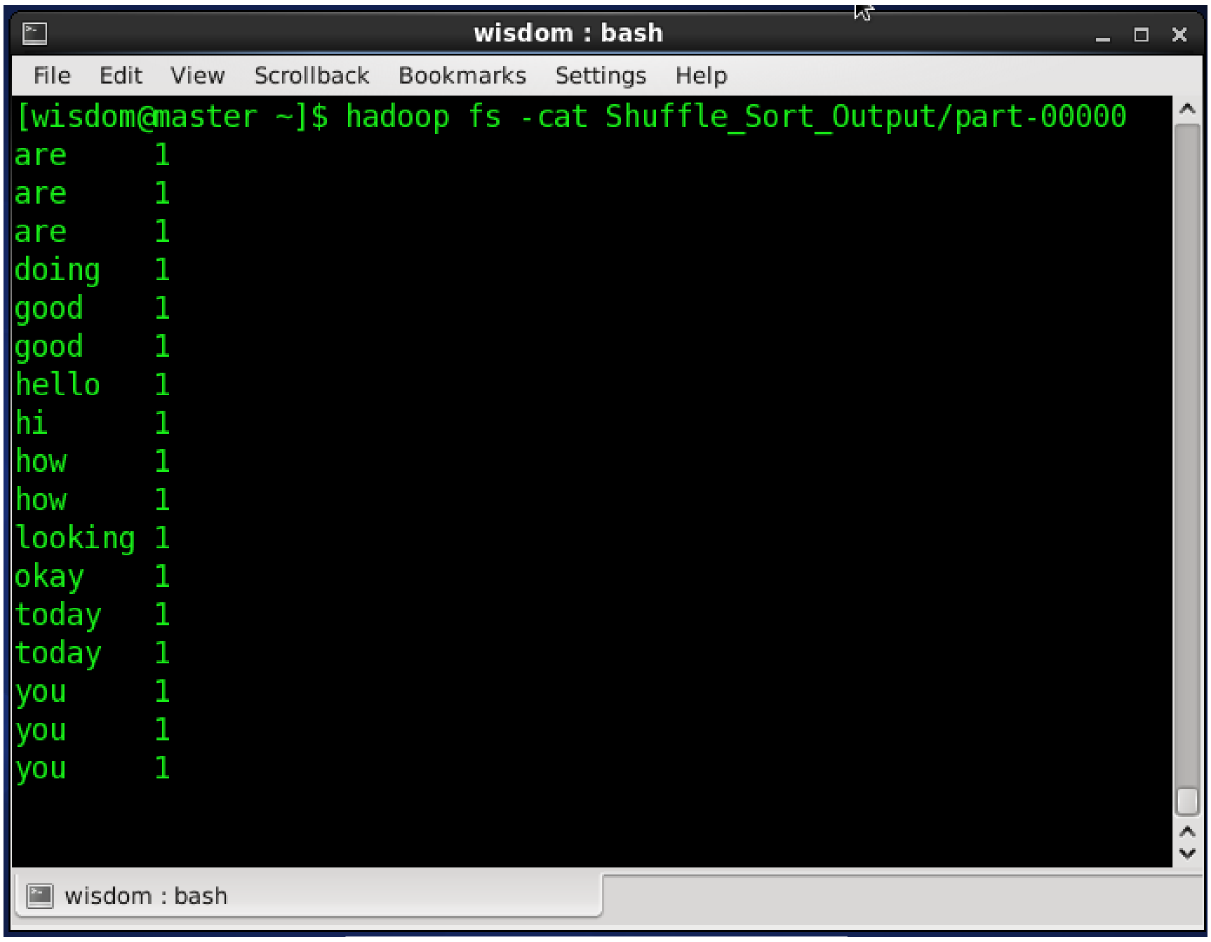
After execution the Mapper will intimate the same to the Job Tracker.

REDUCER IN ACTION

After completion of the Map phase, the Job Tracker will trigger the Reduce Phase. In this phase the Job Tracker will randomly assign the 4th task to any of the available Task Trackers.

This Task Tracker will collect the output files of all the previous 3 tasks, merge them together and execute Reduce function on the merged block. At this point the Task Tracker on duty is called as the Reducer.

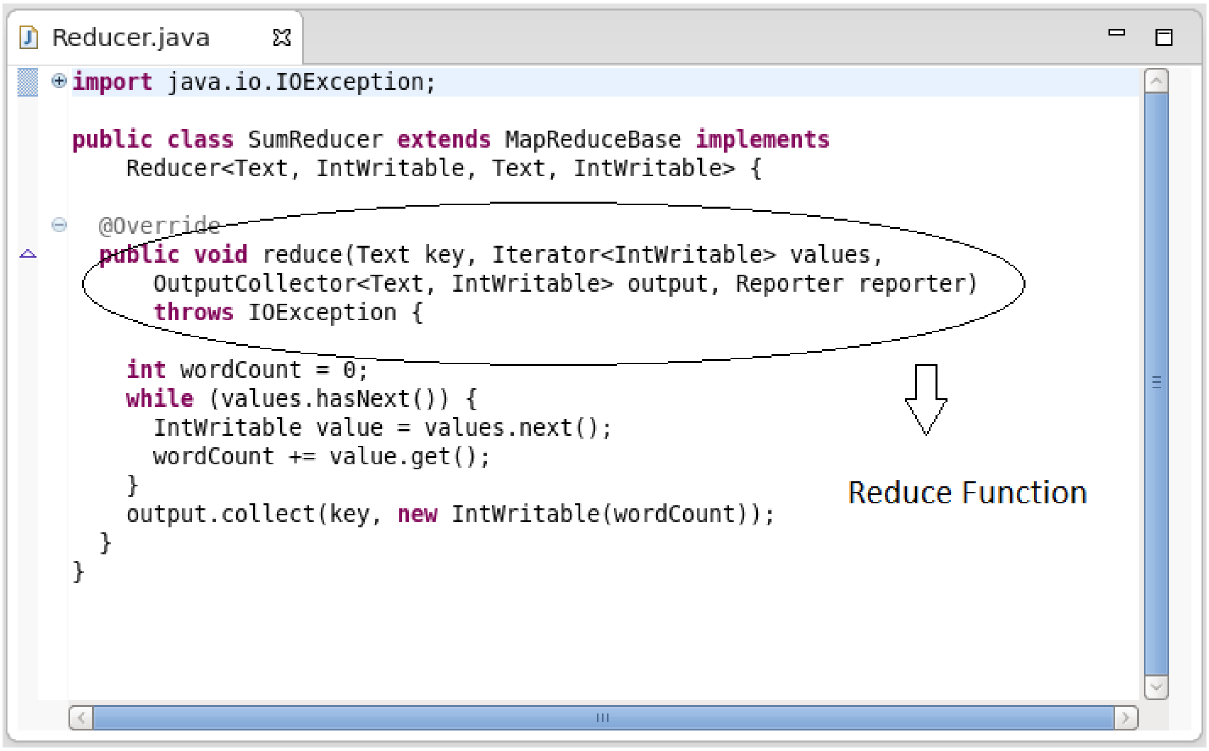
In the Reduce phase, before execution of the user defined function, an internal default function called Shuffle and Sort is executed. The functionality of this phase is fixed. The intent is simple – the keys have been accumulated from all Mappers in a single machine now, so they need to be grouped properly and sorted so as to make counting easy *(The Sorting is done in ASCII format)*.



```
[wisdom@master ~]$ hadoop fs -cat Shuffle_Sort_Output/part-00000
are     1
are     1
are     1
doing   1
good    1
good    1
hello   1
hi      1
how     1
how     1
looking 1
okay    1
today   1
today   1
you     1
you     1
you     1
```

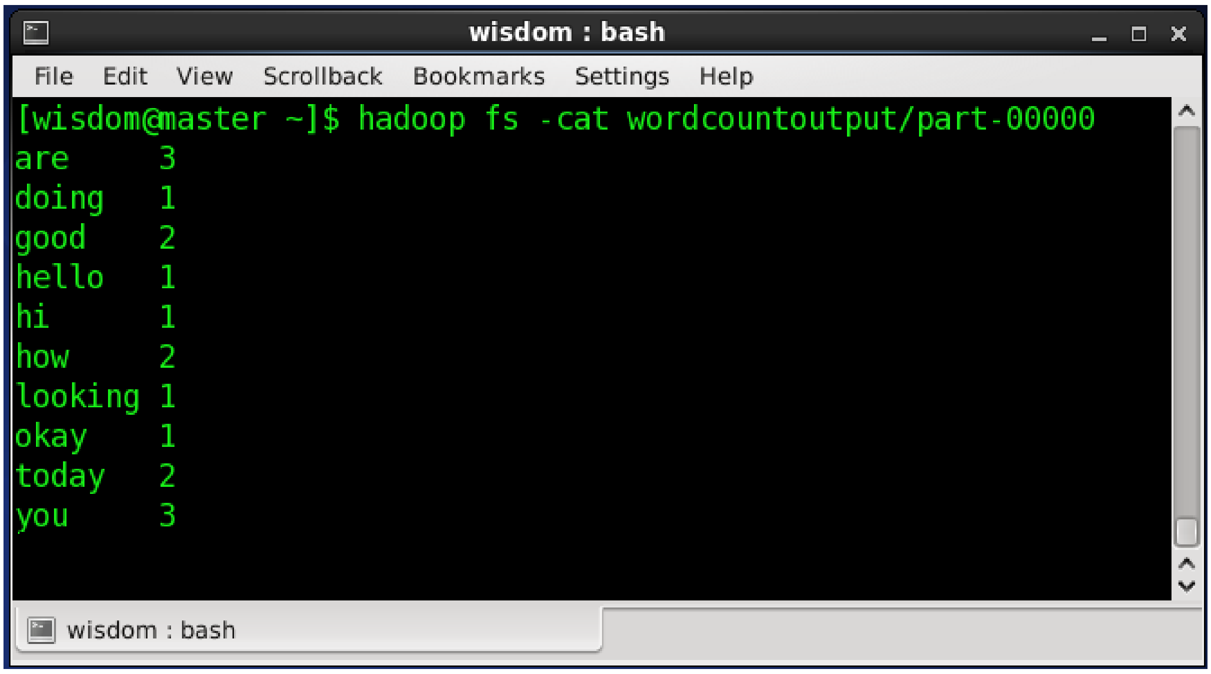
Fig 26: Shuffle and Sort Output

After execution of the Shuffle and Sort, the output is then passed through the Reduce function. Here the Reducer will produce one output file which will be stored in the output folder specified by the client. Additionally two more file will be dumped in this folder viz. *_SUCCESS FILE* => determining the successful execution of the program and *_LOG FILE* => determining the logs related to the program execution. The output file will follow the File Anatomy for the storage.



```
import java.io.IOException;

public class SumReducer extends MapReduceBase implements
    Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterator<IntWritable> values,
      OutputCollector<Text, IntWritable> output, Reporter reporter)
      throws IOException {

    int wordCount = 0;
    while (values.hasNext()) {
      IntWritable value = values.next();
      wordCount += value.get();
    }
    output.collect(key, new IntWritable(wordCount));
  }
}
```

Fig 27: Reduce Function



```
[wisdom@master ~]$ hadoop fs -cat wordcountoutput/part-00000
are     3
doing   1
good    2
hello   1
hi      1
how     2
looking 1
okay    1
today   2
you     3
```

Fig 28: Final Output

After execution of the Reduce function, the Task Tracker will intimate the Job Tracker who will in turn intimate the same to the Client.

This complete process right from the time when the client wanted to store the file till the time when the program got executed on that file is called as Hadoop Job Process.

With this we also complete a basic introduction to the Core Hadoop. In the next chapter we will have a look at the different tools evolving around Hadoop.

HADOOP ECOSYSTEMS

In the last chapter we saw how the basic and only programming model available in Hadoop actually works. But this model had some problems with it when it comes to modelling of real world applications. In this chapter we will run through these disadvantages of Map Reduce and introduce ourselves to its solution.

WHY HADOOP?

Hadoop came into existence because we were not able to handle Big Data. And what we do with Big Data is only analysis. So it is very clear that Hadoop was there only for analysis and not for application development.

HOLE IN THE HEART OF HADOOP CORE

One of the core components of Hadoop was Map Reduce. This programming model was entirely and purely Java based. This made understanding and using Map Reduce a tedious affair.

Moreover, as already seen in the previous chapter this model has a default and compulsory phase in between the Map phase and the Reduce phase viz. Shuffle and Sort. The input and output patterns of this internal phase was fixed and the user was not allowed to change any of it nor even its functionality. This means, though we can write code for Map and Reduce functions, still we will always have to restrict their output and input formats in consideration with the Shuffle and Sort phase. This made it even more difficult to define real world problems in the Map Reduce paradigm.

It was also observed that all real world problems cannot be solved through a single Map Reduce program. Sometimes series of Map Reduce programs were needed to be deployed in order to reach to the final solution. This made life with Hadoop even more difficult. With this the Hadoop developers thought that may be the pre-requisite of Java may actually restrain people from using Hadoop and the problem of Big Data may become even bigger.

MICROSOFT PAINT

What generally happens in MS Paint if you want to draw a square? You simply click on the square icon in the top panel and then drag, drop and create the square on the canvas. Do you really think that this is the complete process that draws a square? The answer is NO!! At the backend your figure selection triggers some computer graphics like command which captures your clicking coordinates and passes their pixel values as arguments to the corresponding command, upon execution of which the square is driven for you.

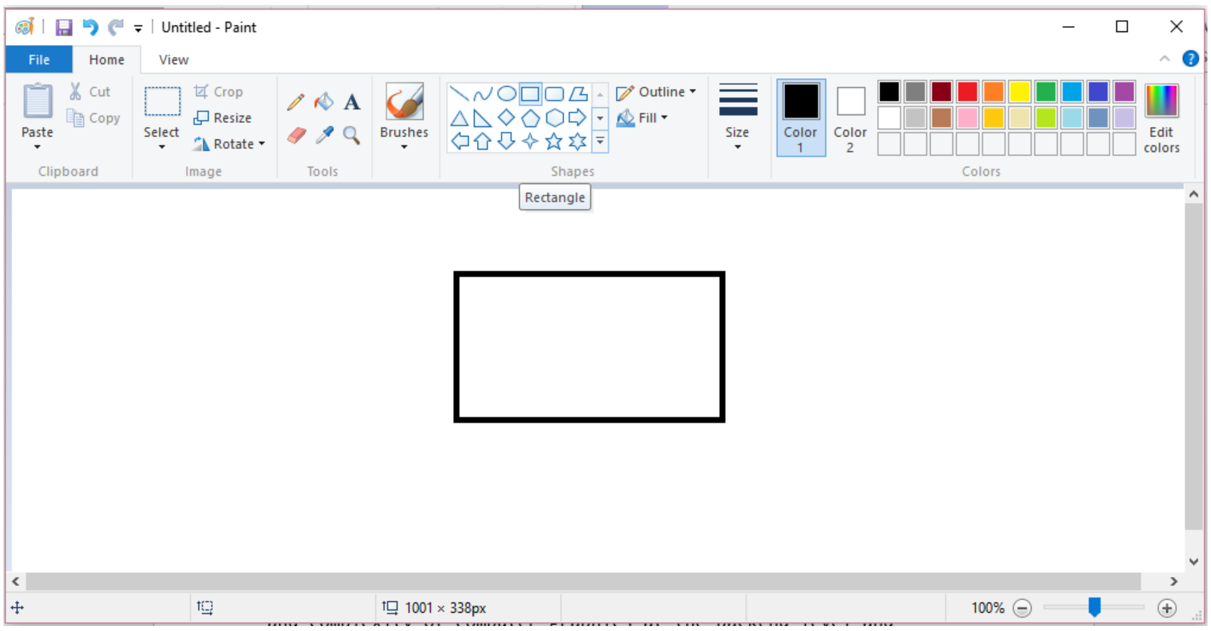


Fig 29: MS Paint

So what have we done in here? We have simply abstracted the details and complexity of computer graphics programs at the backend level and provided a simple user interface at the frontend for ease of use.

HADOOP ECOSYSTEM

On similar grounds the Hadoop developers thought that why not abstract the complexity of Java at the backend level and give some simple English like commands at the frontend. This is where we came out with a number of tools supporting the Hadoop framework or revolving around the Hadoop cluster by executing Map Reduce programs at the backend and giving the analyst a very simple and/or familiar environment on the frontend.

These tools are called as HADOOP ECOSYSTEM Projects. Let's understand certain characteristics of these tools.

1. APPLICATION SPECIFIC

Today I had to cook "Chicken Curry". For this I had to do a lot of preparation out of which collecting some spices and churning them into a paste along with coconut and some other ingredients in a mixer grinder was the most time consuming job. I knew that I would be cooking this type of curry a lot of time in future, so in order to save time I created some extra paste for future use. Now whenever I have to cook curry, I don't have to worry about the paste. But the problem that I noticed was, that I cannot use this paste as "Chutney" for "Dosa or Idli". For that I had to make another paste and this paste was not suitable for preparing curry. But from both these paste what I had done was, reduced the preparation time.

Similarly, companies that used Map Reduce programs to work with several modules of some applications, realized that they would be reusing these modules very often in the future and every time they use these modules they will have to develop these complex Map Reduce programs. Moreover there would be many other companies that may have similar requirements. So to avoid this what they did was bundled all the similar Map Reduce programs together as Library and developed simple English like commands to trigger those library programs from the frontend. They packaged this along with a processor and execution engine as an Open Source Hadoop Ecosystem Tool.

On similar front, many other developers developed different Ecosystem tools to support different applications that required complex Map Reduce programming, resulting in the evolution of a very large Hadoop Ecosystem Family.

2. MAP REDUCE AT THE BACKEND

Another feature of these tools is that all of them have MapReduce at the backend. So if you understand the generic backend functionality then it is easy to work with any tool by just understanding the frontend and application.

3. CLIENT SIDE TOOLS

These tools also give ease of deployment as you can directly install them on the Hadoop Client side without the need of playing with your cluster. Only some minor configurations need to be added in your Cluster so as to integrate them together.

BACKEND WORKING

Let's try to understand the generic backend working of these Ecosystem Tools. Let's assume that our client has a "X Ecosystem tool" installed and now wants to execute its command on the cluster. As soon as the client executed the command, the processor of the tool will look for a corresponding MapReduce program in the library. This program will then be configured by the execution engine in the required Jar format and then this Jar file will be executed on the cluster like any other MapReduce program and the client will get notified on the Output.

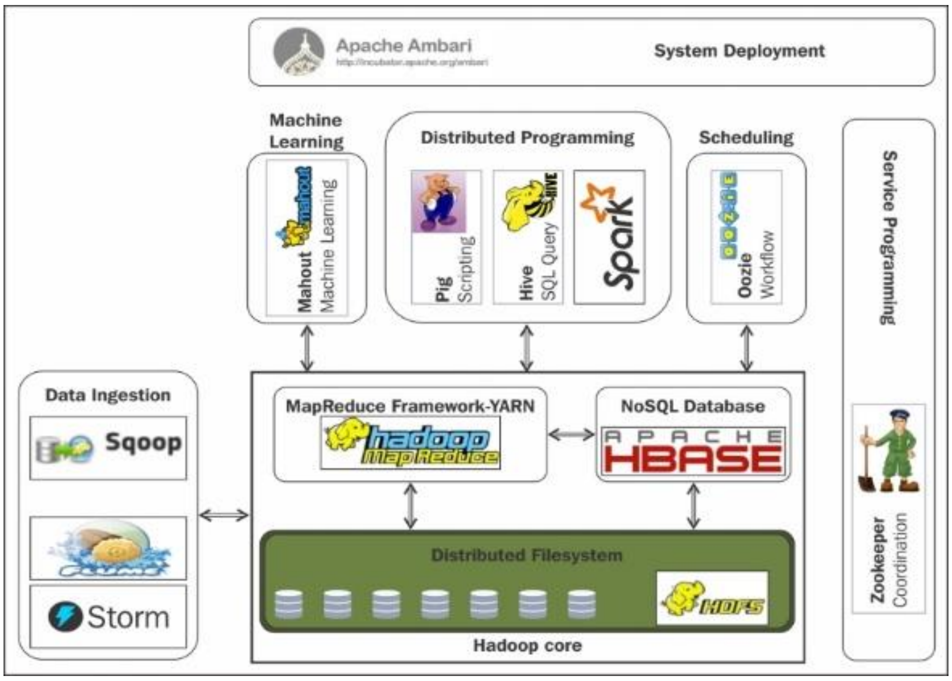
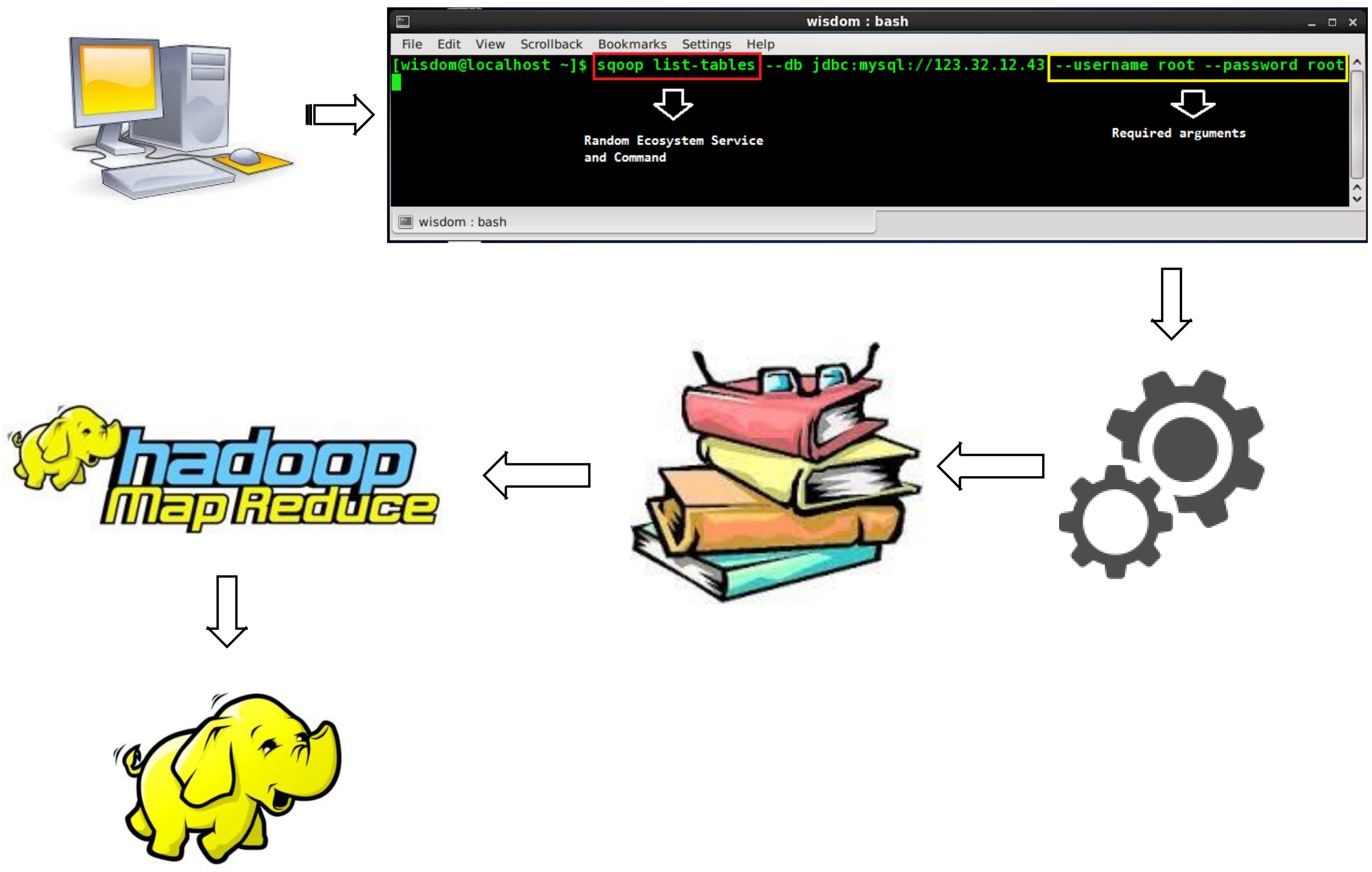


Fig 30: Hadoop Ecosystem



```
[wisdom@localhost ~]$ sqoop list-tables --db jdbc:mysql://123.32.12.43 --username root --password root
```

wisdom : bash
File Edit View Scrollback Bookmarks Settings Help
Random Ecosystem Service and Command
Required arguments
wisdom : bash
hadoop Map Reduce

Fig 31: Ecosystem Tool Backend Job Flow

POPULAR ECOSYSTEM TOOLS

Out of the many Ecosystem tools available the following are the most popular ones that we will study in the next chapters

1. SQOOP

a. This is an Import-Export tool between HDFS and RDBMS

2. HIVE

a. This is SQL on top of Hadoop.

3. PIG

a. This resembles PL/SQL interface

4. FLUME

a. This is a complete simple messaging service

5. OOZIE

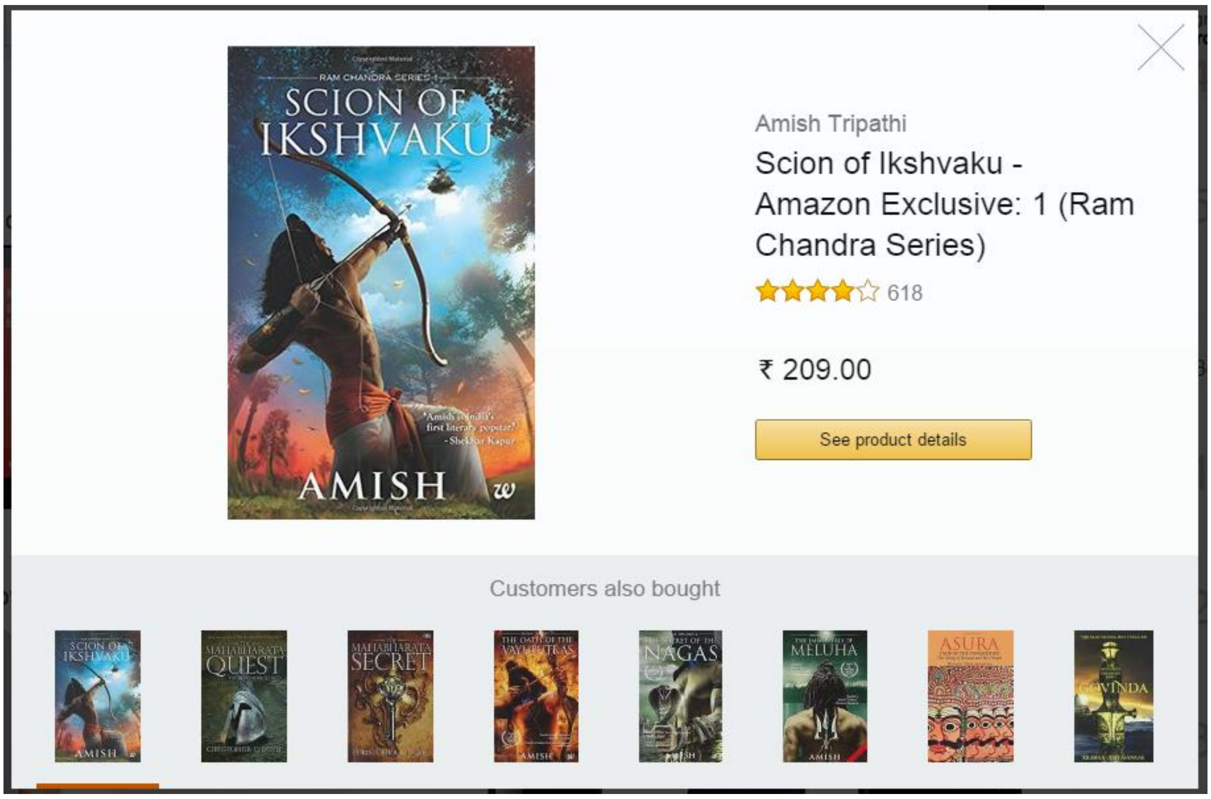
a. This works as a workflow scheduler for Hadoop.

In the next chapter we will have a brief look on the details of one of the Ecosystem tools SQOOP.

INTRODCUTION TO SQOOP

Our Web Architecture is basically divided into 2 sections viz. OLTP *(Online Transaction Processing)* and OLAP *(Online Analytical Processing)*. The transactional data is stored in the databases and the data required for analysis is stored in Data Warehouses *(This is where Hadoop sits)*. So one of the major tasks in the industry deals in transferring the data between the Database and the Hadoop System.

For instance visualize the working of Amazon's recommendation engine.



When you click on a product you get a list of recommended products. So where do we actually get this from? The Databases, right? The databases are connected to the Website, so whenever we click on a product a query asking for the recommended products is fired on the database and the corresponding list is displayed back on the Website. But how do we conclude the recommendations in the database? What we do is, in the Data Warehouse *(Hadoop HDFS)* we collect years of data in terms of mapping between a product and its buyer *(this becomes the product)* and the list of other products that the buyer has brought or searched for *(this becomes the recommendation)*. Then some more analysis is applied and we finally get a list of recommendations for all the products. This list is then uploaded into the Database which fetches the recommendations for us.

One of the traditional approach to handle this requirement is to write MapReduce jobs and get our work done. But this involves a lot of things like establishing a connection, Understanding the Schema, Understanding and defining the Delimiters, Conversion into Text format etc. which makes it a very tedious job. So in order to abstract the complexity and provide a simple interface to work with, Hadoop Ecosystem introduces an Import Export tool on Hadoop called SQOOP.

SQOOP BASIC OPERATIONS

Using Sqoop we can query the database for some basic operations

Syntax:

sqoop list-databases --connect <**communication protocol**>:<**database type**>://<**database server address**>/<**active server schema**> --username <**database username**> --password <**database password**>

This command will connect to the database server specified through the connection string and will validate against the username and password. If this goes well then it will display the list of the databases in that server on the console. In the connection string we will be specifying the communication protocol used, the type of the database server as well as the location of the server on the network.

Syntax:

sqoop list-tables --connect <communication protocol>**:<database type**>://<**database server address**>/<**active server schema**> --username <**database username**> --password <**database password**>

The above command will list all the tables in the said active server schema.

Syntax:

sqoop eval --connect <communication protocol>**:<database type**>://<**database server address**>/<**active server schema**> --query <query to be executed on the database> --username <**database username**> --password <**database password**>

This command will execute the embedded query on the database and will display the result on the console in the Client machine. This command can be used to determine the outcome of a Sqoop Selective Import.

INSIGHTS OF SQOOP

Sqoop is an import export tool between HDFS and RDBMS. Almost all the RDBMS make use of JDBC to communicate with third parties as well as with different database. As Sqoop also uses the same interface for communication, it can definitely communicate with all the RDBMS. The commands for all the databases is the same. You only need to check whether you have the proper drivers installed on your Client machines.

For databases, Sqoop will read the table row-by-row into HDFS. The output of this import process is a set of files containing a copy of the imported table or datasets. The import process is performed in parallel. For this reason, the output will be in multiple files. These files may be delimited text files, or binary Avro or Sequence Files containing serialized record data.

After manipulating the imported records you may have a result data set which you can then export back to the relational database. Sqoop's export process will read a set of delimited text files from HDFS in parallel, parse them into records, and insert them as new rows in a target database table, for consumption by external applications or users.

Most aspects of the import, code generation, and export processes can be customized. For databases, you can control the specific row range or columns imported. You can specify particular delimiters and escape characters for the file-based representation of the data, as well as the file format used.

SQOOP IMPORT

When it comes to importing data from RDBMS to HDFS Sqoop offers different flavours as follows

***The data set used for this section is of the student table, sample of which is available towards the end of the chapter.*

IMPORT SINGLE TABLE

Syntax:

sqoop import --connect <communication protocol>**:<database type>**://<**database server address**>/<**active server schema**> --table <**database table name**> fields-terminated-by <**delimiter to be used**> --username <**database username**> --password <**database password**>

Example Command:

```
sqoop import --connect jdbc:mysql://192.168.1.200/wisdom_db --table
students fields-terminated-by '¥t' --username root --password root
```

This command will create a directory in HDFS by name students and will dump the data of the table students in a tab separated text file in the newly created HDFS directory. Additionally you can specify the name of the directory where you want to dump the output instead of the default location.

```
--target-dir <hdfs directory path>
```

This command basically requires that the table students should have a primary key field associated in the database. The default MapReduce program for this operation has certain characteristics as follows

IT IS A ONLY MAPPER MAPREDUCE PROGRAM

As we are not doing any aggregation and as we are just copying the data from the database and dumping it as it is in to the HDFS, we don't need the Reduce phase.

IT CONSISTS OF 4 MAPPERS BY DEFAULT.

Every execution of the Map function will by default fetch 25% of the total data available and dump it in to the HDFS.

So in order to split the data among the 4 Mappers, the primary key is essential. If the table does not contain the primary key then, you will have to specify an additional operation containing the details of the field in the table which can be considered for creating the splits.

```
--split by <column name>
```

If it is not possible to select a field for performing the split then you also have an option of using only one Mapper for importing the entire 100% data. For this you will have to specify an additional option stating that the number of Mappers to be used for this operation should be 1.

```
--m 1
```

IMPORT RESULT SET

Sqoop also permits you to run a query on the database and sump the result set of this operation into the HDFS.

```
Example Command:
sqoop import --connect jdbc:mysql://192.168.1.200/wisdom_db --table
students fields-terminated-by '¥t'--where marks>75 --target-dir
my_result --m 1 --username root --password root
```

This command will execute a select query of the following type on the database table students and the result set will be dumped in the target directory.

```
Sample database query:
select * from students where marks>75;
```

INCREMENTAL IMPORT

It may happen that you have already imported data of a table and now you need to check for additional data entry and import only the newly added records. For doing this you can add up the following options in your import command.

```
--check <name of the field>
```

This will check on addition in the records with respect to the filed specified.

```
--incremental <mode>
```

You can specify an append mode wherein the filed will be checked for newly added records or you can specify the last-modified mode wherein the field will be checked for newly updated records

```
--last-value <value or time-stamp>
```

This option will specify the last known value of the field if the append mode is used. In case of the last-modified mode, this option will specify the last known time-stamp.

IMPORT ALL TABLES

In order to import the data from all the tables in the database you can use the following command:

```
Example Command:
sqoop import-all-tables --connect
jdbc:mysql://192.168.1.200/wisdom_db --username root --password
root
```

This command will import all tables in the database and will dump it in HDFS under directories having the names corresponding to the table names in the database. This command strictly requires that every table in the database has a Primary Key field.

SQOOP EXPORT

Now, Sqoop also facilitates us with a way of exporting data from the HDFS and dumping it into a table inside the database server.

Syntax:

sqoop export --connect <communication protocol>**:<database type>**://<**database server address**>/<**active server schema**> --table <**name of target table in the database**> --username <**database username**> --password <**database password**> --export-dir <**hdfs directory where the data to be exported is present**>

Example Command:

```
sqoop export --connect jdbc:mysql://192.168.1.200/wisdom_db --table
attendance --username root --password root --export-dir
/user/wisdom/student_attendance.txt –fields-terminated-by '¥t'
```

For this command to execute successfully it is required that we have a table in the database where we are trying to dump the data from HDFS.

In this way Sqoop can be used under different flavours to exchange data between HDFS and RDBMS without actually writing a single line of Java MapReduce code.

DATA SETS USED

Students Table

Id	Name	Grade	Marks
1	Mahesh	2	56
2	Ramesh	2	87
3	Satish	2	78
4	Rohan	2	42
5	Mary	2	50
6	Simran	2	66

Schema of Attendance Table

Id	Name	Total Days	Days Present	Days Absent

student_attendance.txt

```
1    Mahesh  30   24   6
2    Ramesh  30   22   8
3    Satish  30   26   4
4    Rohan   30   20   10
5    Mary    30   28   2
6    Simran  30   12   18
```

INTRODUCTION TO HIVE

In this chapter we will have a brief look at the Hadoop Ecosystem tool called as HIVE.

WHAT IS HIVE?

Talking about analysis the main task always remain is Data Mining. Data Mining means deriving useful information from large chunks of data. The simplest way of doing this has always been asking question to the data. This was a very easy task when we were dealing with the database. Using the Structured Query Language *(SQL)* we were able to query the data easily.

When it came to Hadoop, this task became tedious as Hadoop does not support SQL. We had to write complex Map Reduce programs to do those functions that were once done by simple SELECT queries. In order to ease this task we came up with a tool that worked as a Map Reduce SQL abstraction or SQL on top of Hadoop. This tool is called as HIVE.

When HIVE is installed on the client machine we get a HIVE metastore that is configured on the client side and on the cluster we need to build a HIVE warehouse.

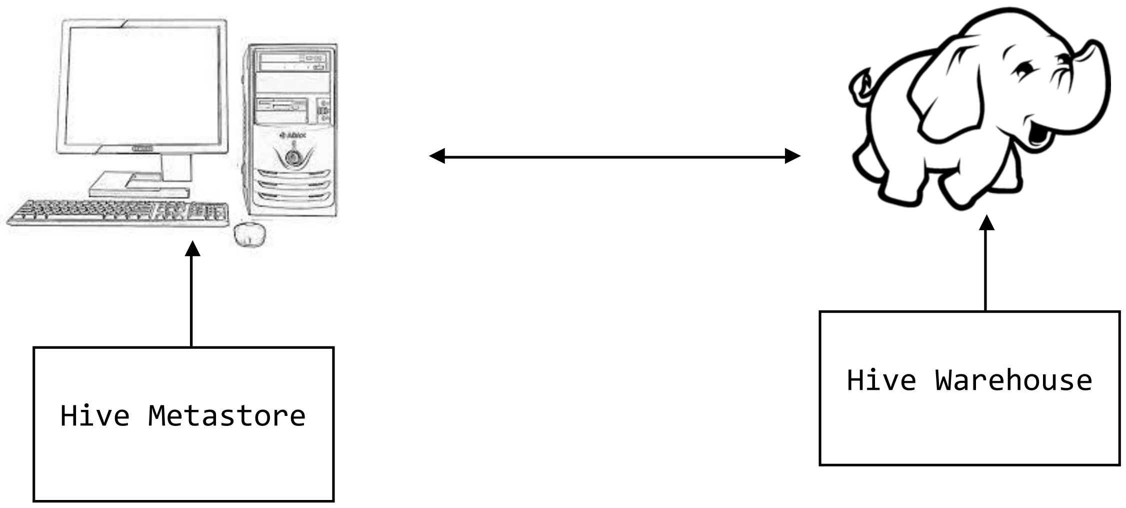


Fig 45: HIVE Metastore and Warehouse

The HIVE metastore is a place where the HIVE query engine keeps all the Schema related information or the so called Meta Data. The HIVE warehouse inside the HDFS is where the actual HIVE table data is stored. You can easily visualize the warehouse as a part of HDFS distributed across all the Data Nodes.

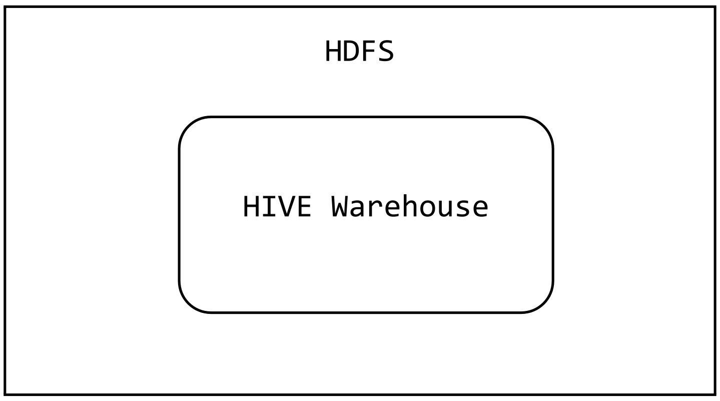
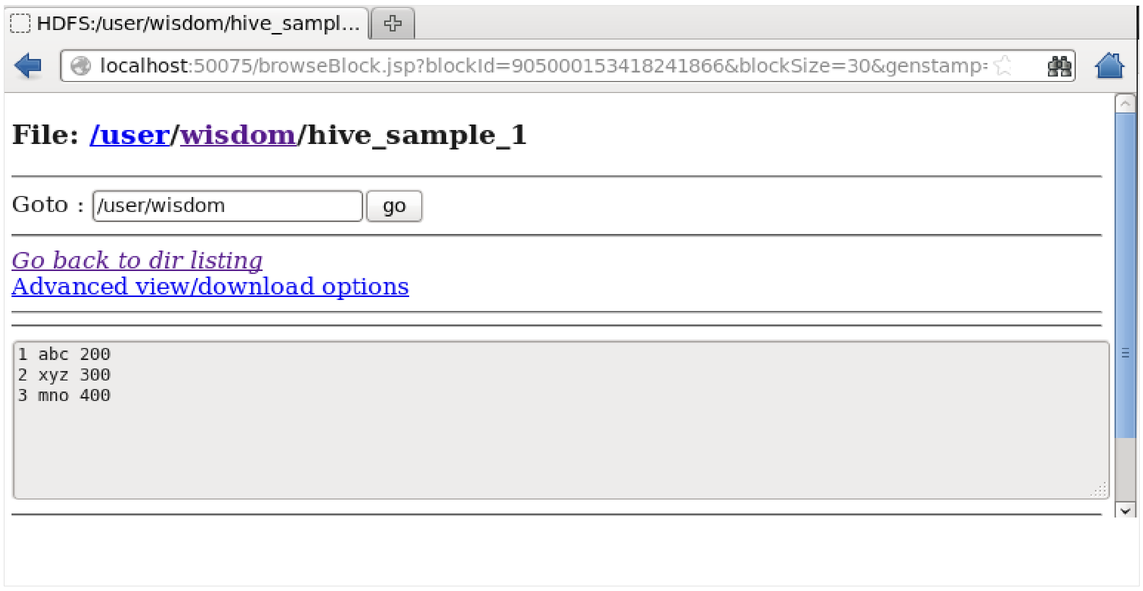


Fig 46: HDFS and HIVE Warehouse

HIVE – SQL ON TOP OF HADOOP

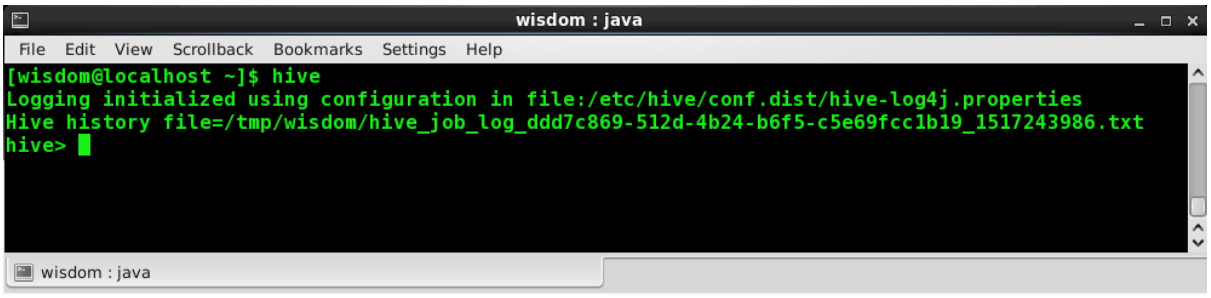
Let's assume we have a tab separated file in HDFS that we want to query for some sort of data analysis.


HDFS:/user/wisdom/hive_sampl...
localhost:50075/browseBlock.jsp?blockId=905000153418241866&blockSize=30&genstamp=
File: /user/wisdom/hive_sample_1
Goto : /user/wisdom go
Go back to dir listing
Advanced view/download options

```
1 abc 200
2 xyz 300
3 mno 400
```

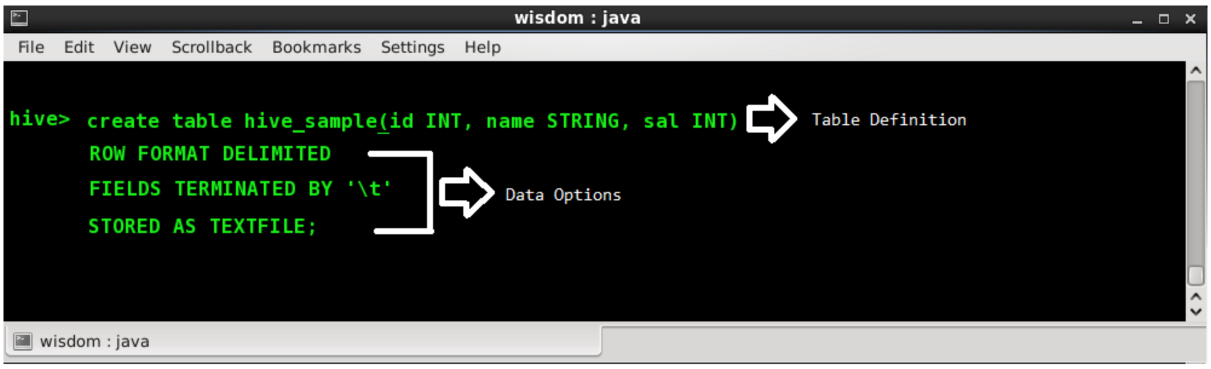
Fig 47: Sample data in HDFS

What we do is, using the HIVE tool we will first create a table that will resemble the schema of the data in that file.



```
[wisdom@localhost ~]$ hive
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
Hive history file=/tmp/wisdom/hive_job_log_ddd7c869-512d-4b24-b6f5-c5e69fcc1b19_1517243986.txt
hive>
```

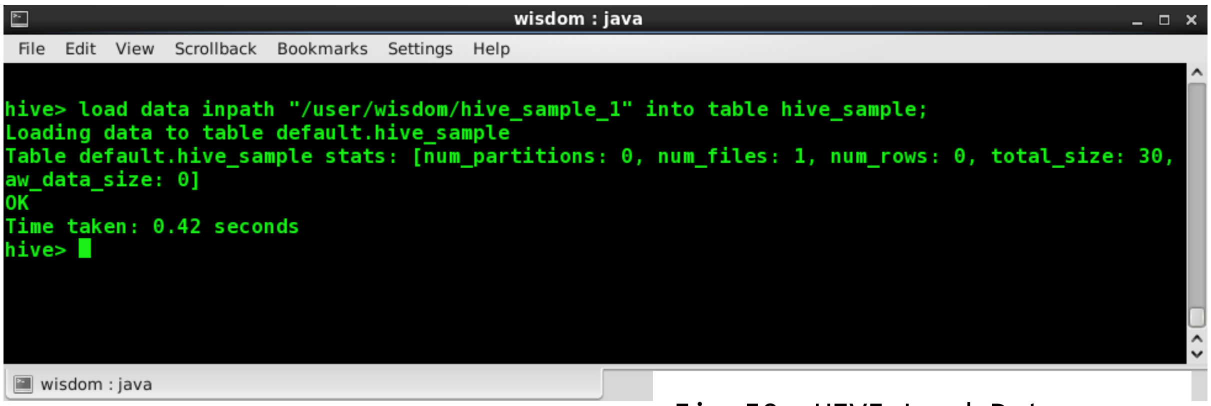
Fig 48: HIVE Shell



```
hive> create table hive_sample(id INT, name STRING, sal INT)
      ROW FORMAT DELIMITED
      FIELDS TERMINATED BY '\t'
      STORED AS TEXTFILE;
```

Fig 49: HIVE Create Table

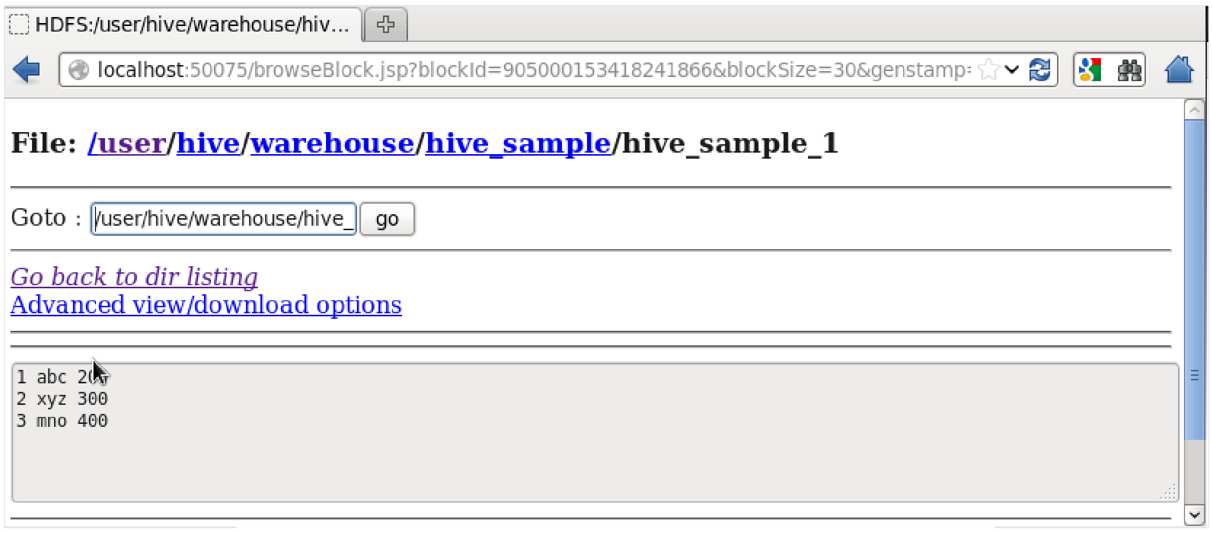
This schema is stored by the HIVE query engine into the HIVE metastore. Normally when you create a table in any traditional database, what happens is that a blank table is created where you can then insert some data. But this does not happen in HIVE. Rather in HIVE only the definition is stored and no other action on that is taken. Then when you issue a command to load data from a file in HDFS into this newly created table *(the definition)*, data is actually not loaded anywhere.



```
hive> load data inpath "/user/wisdom/hive_sample_1" into table hive_sample;
Loading data to table default.hive_sample
Table default.hive_sample stats: [num_partitions: 0, num_files: 1, num_rows: 0, total_size: 30,
aw_data_size: 0]
OK
Time taken: 0.42 seconds
hive>
```

Fig 50: HIVE Load Data

What happens next is, that the data file is moved from the HDFS to the HIVE's warehouse and a link is established between the metadata for the table and this file.


HDFS:/user/hive/warehouse/hiv...
localhost:50075/browseBlock.jsp?blockId=905000153418241866&blockSize=30&genstamp

File: /user/hive/warehouse/hive_sample/hive_sample_1

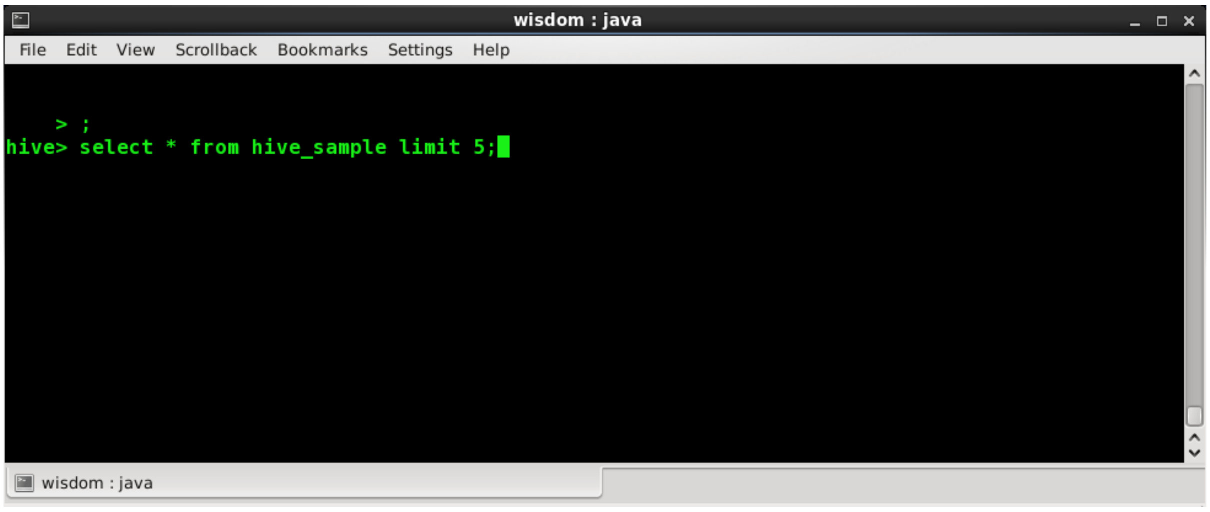
Goto : /user/hive/warehouse/hive_ go

Go back to dir listing
Advanced view/download options

```
1 abc 200
2 xyz 300
3 mno 400
```

Fig 51: HIVE data in moved in Warehouse

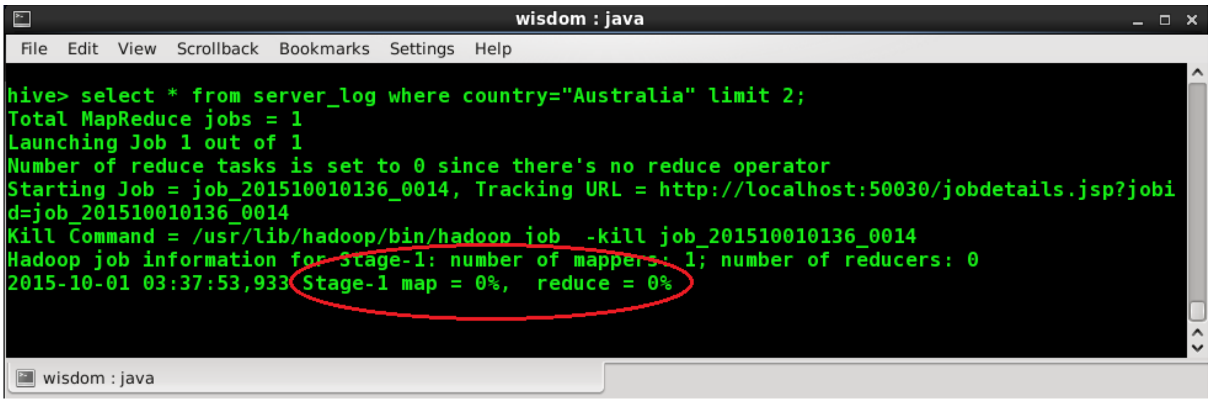
Henceforth the metadata and the file are always referenced as schema and data of each other. So now if you execute a SELECT query on that table, the HIVE engine will fetch a relevant Map Reduce program from the HIVE library and execute it on the data file present inside the warehouse.



```
    > ;
hive> select * from hive_sample limit 5;
```

Fig 52: HIVE Query

While execution as well as while displaying the output, the schema for that table from the metastore will be taken into consideration and the data will be displayed as if it was directly fetched from a database table using SQL queries. But what we are actually doing is simply executing Map Reduce programs at the backend and the SQL on the front. Hence HIVE is also called as SQL on top of Hadoop.



```
hive> select * from server_log where country="Australia" limit 2;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201510010136_0014, Tracking URL = http://localhost:50030/jobdetails.jsp?jobi
d=job_201510010136_0014
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_201510010136_0014
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2015-10-01 03:37:53,933 Stage-1 map = 0%,  reduce = 0%
```

Fig 53: HIVE Map Reduce at the Backend

HIVE DATA TYPES

Hive supports different categories of Data types as follows

INTEGER

The field can be specified to have Integer as a data type by using the keyword INT against the field name. If the value that the field will contain is supposed to be beyond the range of an Integer then Big Integer *(BIGINT)* can be used whereas if the value is supposed to be below the range then Small Integer *(SMALLINT)* can be used or if the value is supposed to be smaller than the Small Integer, the Tiny Integer can also be used.

STRING

String type data types can be specified using single quotes *(' ')* or double quotes *(" ")*. It contains two data types: VARCHAR *(length: 1 to 65535)* and CHAR *(length: 255)*.

TIME STAMP

It supports traditional UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format YYYY-MM-DD HH:MM:SS.fffffffff and format yyyy-mm-dd hh:mm:ss.ffffffffff.

DATE

DATE values are described in year/month/day format in the form {{YYYY-MM-DD}}.

FLOATING POINT TYPES

Floating point types are nothing but numbers with decimal points. Generally, this type of data is composed of DOUBLE data type.

DECIMAL TYPE

Decimal type data is nothing but floating point value with higher range than DOUBLE data type. The range of decimal type is approximately -10-308 to 10308.

BASIC HIVE OPERATION

If you want to check the version of Hive that you are currently using then you can execute the following command.

```
set system:sun.java.command;
```

If you want to list all the databases currently present in your Hive server then you can use the following command.

```
SHOW DATABASES;
```

You can also use Regular Expression in order to list only the required databases as in the following command.

```
SHOW DATABASES LIKE 'h.*';
```

This command will list all the databases having names starting with the letter h.

By using the following command you can create a new database and use it for performing other tasks

```
CREATE DATABASE IF NOT EXISTS <database name>;
```

With the help of the following command, you can check all the details of a database as in its location etc.

```
DESCRIBE DATABASE <database name>;
```

The following command can be used to create the said database. After creation, tables can be created under it and queries can be execute. The same command is also used to switch to the said database.

```
USE <database name>;
```

The major difference between a traditional database and Hive is the presentation of the tables. In a traditional database you will get a fancy table whereas in Hive you will get a delimited output. To make this output more understandable it is sometimes necessary to display the column names along with the actual data. The following command will print the column names for all further command's output.

```
set hive.cli.print.current.db=true;
```

The following command will delete the said database and all its related components from the Hive repository on your HDFS.

```
DROP DATABASE IF EXISTS <database name> CASCADE;
```

In this way we can enjoy the features of a traditional database on the distributed data storage of Hadoop without much complexity. This is comparatively slower than the traditional databases but that's ok as long as we are concerned only about data analysis. The advantage is that we can now work on a very large scale of data in a single go

In the next chapter we will have a look at the programming version of HIVE i.e. PIG.

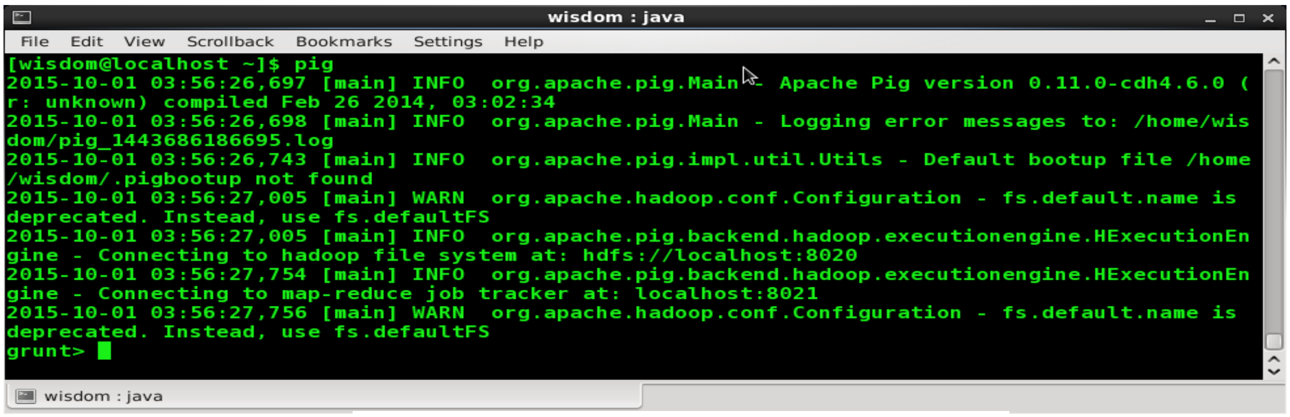
INTRODUCTION TO PIG

If you have some data of a Car Showroom and if I tell you to calculate the number of eligible people that have applied for a loan to buy a Ferrari in the year 2015, by evaluating certain criteria like, if the buyer is a male then, check his wife's Pan Card details if he is married else check his mother's Pan Card details. If any of them (Mother, Wife) falls under the Second Taxation Slab then calculate the minimum EMI that he can pay and if the EMI is below 2,00,000 INR then check the current Salary of that person, if it is more than 3,00,000 INR, then mark him as eligible else discard his application. And do the same kind of calculation if the buyer is a woman.

Would you be able to do this using SQL? The answer simply is a big NO. Of course if you are a SQL expert then you can definitely do it but still the query will have many Joins and Nested queries which will make the query very slow and make this approach unfeasible. So how do we do achieve this then? We have a simple approach of doing this which is an alternate to or the advanced version of SQL i.e. PL/SQL. By using PL/SQL we can easily use the conditional and looping statements and solve our problem.

Similarly if there are any operations that cannot be solved using HIVE or some tasks that may turn out to be tedious if done using HIVE then to make life easy, we have an interface of PL/SQL on Hadoop, and the topic of this chapter that is PIG.

INTRODUCTION



```
[wisdom@localhost ~]$ pig
2015-10-01 03:56:26,697 [main] INFO  org.apache.pig.Main - Apache Pig version 0.11.0-cdh4.6.0 (
r: unknown) compiled Feb 26 2014, 03:02:34
2015-10-01 03:56:26,698 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/wis
dom/pig_1443686186695.log
2015-10-01 03:56:26,743 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home
/wisdom/.pigbootup not found
2015-10-01 03:56:27,005 [main] WARN  org.apache.hadoop.conf.Configuration - fs.default.name is
deprecated. Instead, use fs.defaultFS
2015-10-01 03:56:27,005 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEn
gine - Connecting to hadoop file system at: hdfs://localhost:8020
2015-10-01 03:56:27,754 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEn
gine - Connecting to map-reduce job tracker at: localhost:8021
2015-10-01 03:56:27,756 [main] WARN  org.apache.hadoop.conf.Configuration - fs.default.name is
deprecated. Instead, use fs.defaultFS
grunt>
```

Fig 54: PIG Grunt Shell

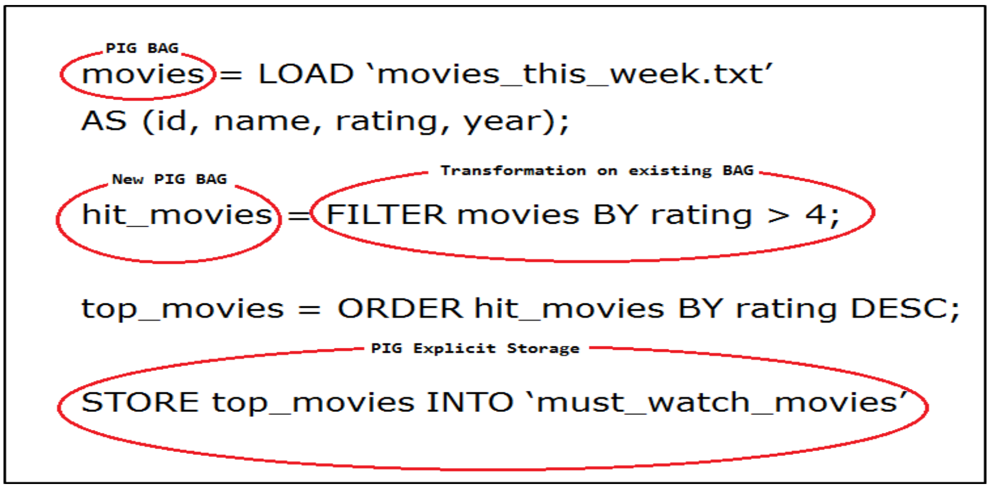
PIG is referred to as a PL/SQL interface on top of Hadoop. We can use PIG and implement different programming aspects like loops and conditions while working out with our tabular data.

PIG is a Scripting Language that can explore huge data sets of size TB and PB easily. It provides a higher level language called as PIG Latin which can be executed in the PIG shell called Grunt.

Unlike HIVE, PIG stores the data and the schema of the data in a single unit at the same place inside the HDFS. In PIG, a Single data element is called as atom, collection of atoms is called as tuple, and collection of tuples is called as BAG. Typically, PIG would load a dataset in a BAG and then create new *BAG* by modifying the existing one.

In the example below, the *movies_this_week* file is loaded in *movies BAG*, then a new *BAG* called *hit_movies* is created for movies with ratings more than 4. Later a new *BAG top_movies* is created and finally the results are stored in a file called *must_watch_movies*.

Every time we process the data inside a *BAG* a new *BAG* is created, as a result of the operation. In order to avoid the unnecessary additional replication of data, which results out of this *BAG* creation of PIG, it does not implicitly store the *BAGs*. A *BAG* is alive only for the current session in which it is created. If the session is terminated, the *BAG* is lost. In order to store the *BAG* for future use, we need to explicitly store that *BAG*.



```
movies = LOAD 'movies_this_week.txt'
AS (id, name, rating, year);

hit_movies = FILTER movies BY rating > 4;

top_movies = ORDER hit_movies BY rating DESC;

STORE top_movies INTO 'must_watch_movies'
```

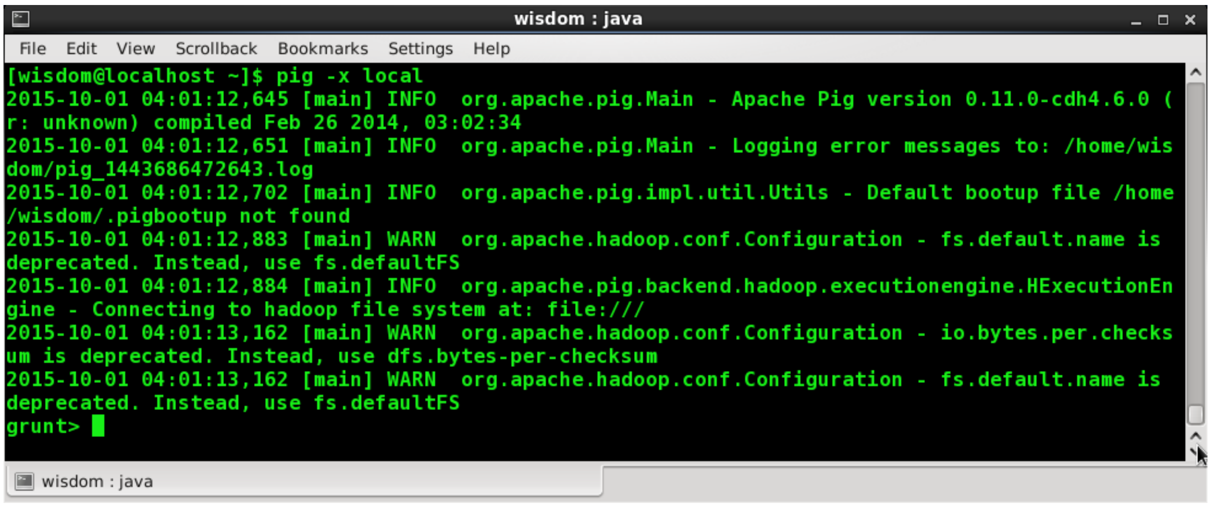
Fig 55: Elements of PIG

PIG EXECUTION MODES

PIG can be operated in 2 modes viz. Map Reduce mode and Local mode.

The Map Reduce Mode is the default mode of PIG. In this mode the input is collected from HDFS and the output is dumped back to the HDFS.

In the local mode the input and output are from the local file system. This mode of PIG is useful for evaluating samples of the actual input to understand the pattern of the data.



```
[wisdom@localhost ~]$ pig -x local
2015-10-01 04:01:12,645 [main] INFO  org.apache.pig.Main - Apache Pig version 0.11.0-cdh4.6.0 (
r: unknown) compiled Feb 26 2014, 03:02:34
2015-10-01 04:01:12,651 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/wis
dom/pig_1443686472643.log
2015-10-01 04:01:12,702 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home
/wisdom/.pigbootup not found
2015-10-01 04:01:12,883 [main] WARN  org.apache.hadoop.conf.Configuration - fs.default.name is
deprecated. Instead, use fs.defaultFS
2015-10-01 04:01:12,884 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEn
gine - Connecting to hadoop file system at: file:///
2015-10-01 04:01:13,162 [main] WARN  org.apache.hadoop.conf.Configuration - io.bytes.per.checks
um is deprecated. Instead, use dfs.bytes-per-checksum
2015-10-01 04:01:13,162 [main] WARN  org.apache.hadoop.conf.Configuration - fs.default.name is
deprecated. Instead, use fs.defaultFS
grunt>
```

Fig 56: PIG Shell in local mode

DATA TYPES IN PIG

Data Type	*Description*
Int	32 Bit Integer
Long	64 Bit Integer
Float	32 Bit Floating Point Number
Double	64 Bit Floating Point Number
Chararray	String
Bytearray	Blob

BASIC TRANSFORMATIONS IN PIG

Operator	*Description*
Filter	Selects a set of tuples from a relation based on a condition.
For Each	Iterates the tuples of a relation, generating a data transformation.
Group	Groups the data in one or more relations.
Join	Joins two or more relations
Load	Loads data from the file system.
Order	sorts a relation based on one or more fields
Split	Partitions a relation into two or more relations.
Store	Stores data in the file system.

PIG OPERATIONS

LOADING DATA

Syntax:

<**Pig Bag Name**> = Load '<**hdfs file location**>' using PigStorage ('<**type of delimiter**>');

This command will load the data of the file in to the said Pig Bag. If the mode of Pig Operation is local then the file location should be from the local file system, else it should be the Hdfs location. By default Pig delimits the file using tabs, but you can specify your own delimiter by using the PigStorage option.

Example:

```
employee = Load '/user/wisdom/employee_data.csv' using PigStorage
(',');
```

You can also specify the definition or schema for the bag using the AS keyword

Syntax:

AS (<**Field 1 Name**> :< **Field 1 Data Type**>, <**Field 2 Name**>:<**Field 2 Data Type**>, <**Field n Name**>:<**Field n Data Type**>)

Example:

```
employee = Load '/user/wisdom/employee_data.csv' AS (id: int, name:
chararray, salary:int, position chararray) using PigStorage (',');
```

FILTER COMMAND

One of the major roles in data analysis is to refine the data set by applying certain conditions on to the original data set. By using the Pig command of Filter, this can be achieved easily.

Syntax:

<**pig bag**> = FILTER <**existing bag**> BY <**field name**><**filter condition**>

```
Example:
list_of_managers = FILTER employee BY position == manager
```

The above command will filter all data from the existing bag employee based on the condition that the position of the employee should be manager. Pig will then create a new bag called list_of_managers out of the result set.

PIG GROUPING

Just like any other traditional database, even Pig provides the alternative to functionally group the subsets of your data together based on certain common properties.

Syntax:

<**pig bag**> = GROUP <**existing bag**> BY <**field name**>

```
Example:
company_map = GROUP employee BY position;
```

PIG SAMPLING

Even if you have more than Brontobytes of data, using Hadoop it can be easily analyzed. But before we could analyze it, it is necessary for the analyst to understand what is there in the data and what does he need out of it. For this the data has to be initially manually analyzed. Now the problems is, how to manually understand such large amount of data? Instead of evaluating the entire data set what we do is, evaluate only a part of it. For this purpose Pig provides an option of sampling the data.

Syntax:

<**pig bag**> = SAMPLE <**existing bag**> <**sample ratio**>

Example:

```
employee_sample = SAMPLE employee 0.01;
```

In the above example, the bag employee is split and a random portion of the bag is returned as the new employee_sample bag.

PIG SPLIT

Pig Split is a perfect example of a MapReduce using more than one Reducer. You can create multiple small bags from the parent bag based on certain conditions.

Syntax:

SPLIT <**existing bag**> INTO <**new bag 1**> IF <**condition**>, <**new bag 2**> IF <**condition**>;

Example:

```
SPLIT employee INTO manager IF position == manager, developer IF
position == developer, analyst IF position == analyst;
```

In this example the bag employee will be split into 3 bags based on the position of each employee in the company.

DESCRIBING A BAG

In case you are not sure about the schema of a Bag, then you can use the Pig Describe method and it will display the structure of the bag on the console.

```
Syntax:
DESCRIBE <Pig bag name>;
```

In this way by using Pig you can easily transform your data using conditions and loops. This was a brief introduction to PIG. In the next chapter we will have a look in to a messaging tool of Hadoop Ecosystem called FLUME.

INTRODUCTION TO FLUME

One of the most important component of the Web Architecture is the Web Server. Not only because it provides service for our Web Site but also because we can fetch a lot of useful information out of it. Information like number of online users, pages most visited, pages least visited, time slot of more user traffic, number of error, etc. This can be further used to do a lot of analysis for deriving business information.

For example, if an e-commerce company has introduced a new module on their Web Site and want to track the popularity and efficiency of that module. In this case they are required to constantly monitor the user activity on their Website. So how can they achieve this?

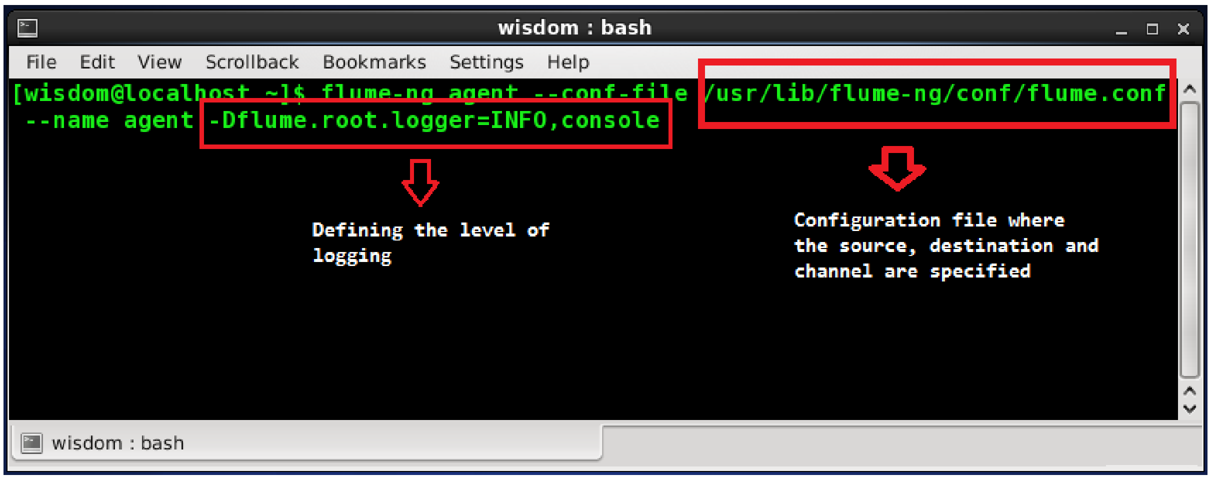
The answer is simple. What they need to do is analyse their Web Server logs and from that they would be able to track the user activity as well as trace any error generated. But now the challenge is that, the logs on the server is an ever incrementing file and if we talk about companies like Amazon, then their Web Servers generate around PB of data almost every hour. So how to analyse such large amount of data.

As a solution to this problem we have a Hadoop Ecosystem tool that acts like a messenger and will keep a track of all updates happening in the logs file and will reliably dump those into the HDFS. Then using tools like HIVE and PIG you can easily analyse it and perform your desired calculations.

This tool is called as FLUME and this is what we would be talking around in this chapter.

WHAT IS FLUME

So what is FLUME exactly all about? FLUME is a simple messaging service that can monitor any ever incrementing file and dump the updates in the HDFS. FLUME is generally a separate server altogether and is neither installed on Client nor on the Cluster. One instance of the FLUME server is called as a FLUME Agent. One Flume agent can be used to monitor a single ever incrementing file. All you need to do is, tell the FLUME Agent the location of the Source file which it has to monitor, the destination directory in HDFS where it has to dump the updates and the communication channel that is to be used for this operation. Once you configure and start the FLUME agent, this continuous service will keep on monitoring the changes in your source file and will keep on dumping it in the HDFS. Optionally you can also configure the time interval after which the updates are to be dumped. BY default it will dump the updates as and when they happen.

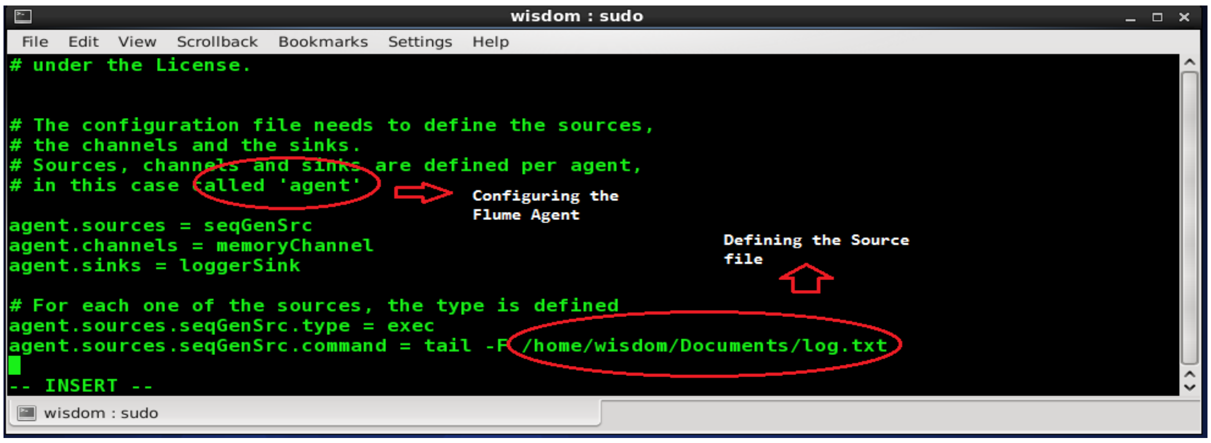

wisdom : bash
File Edit View Scrollback Bookmarks Settings Help
Defining the level of logging
Configuration file where the source, destination and channel are specified
wisdom : bash

```
[wisdom@localhost ~]$ flume-ng agent --conf-file /usr/lib/flume-ng/conf/flume.conf
 --name agent -Dflume.root.logger=INFO,console
```

Fig 57: Starting the FLUME Agent

WORKING OF FLUME AGENT

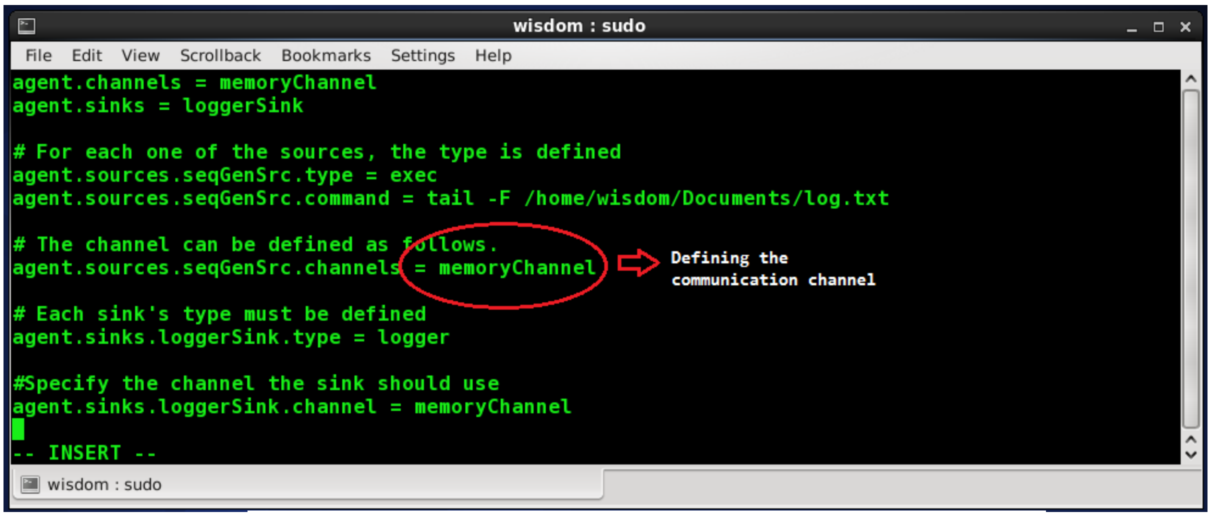
So now let's assume that we have a Web Server log file that is to be analysed. For this reason what we will do is configure the FLUME agent and give it a destination directory inside the HDFS to dump the updates on the log file.


wisdom : sudo
File Edit View Scrollback Bookmarks Settings Help
Configuring the Flume Agent
Defining the Source file
wisdom : sudo

```
# under the License.


# The configuration file needs to define the sources,
# the channels and the sinks.
# Sources, channels and sinks are defined per agent,
# in this case called 'agent'

agent.sources = seqGenSrc
agent.channels = memoryChannel
agent.sinks = loggerSink

# For each one of the sources, the type is defined
agent.sources.seqGenSrc.type = exec
agent.sources.seqGenSrc.command = tail -F /home/wisdom/Documents/log.txt

-- INSERT --
```

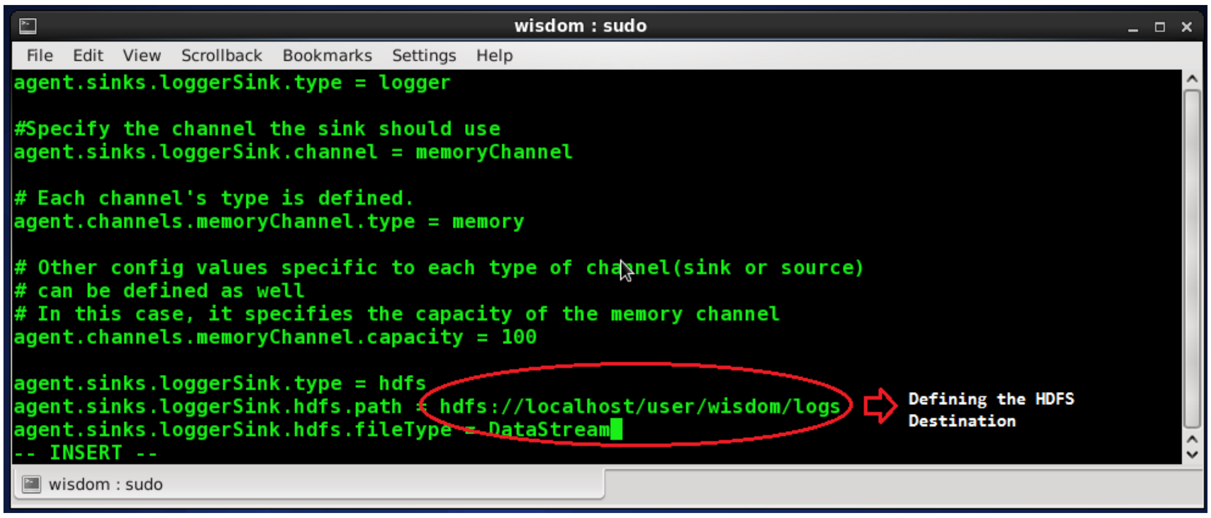
Fig 58: Defining the Source

Once the FLUME agent is started it will start monitoring the log file. Whenever any updates take place it will create something called as events out of it. We can visualize events as a temporary file. The Agent will keep on doing this for a specified interval of time, after which it will bundle all the events together and dump it in the HDFS. By the time this happens there are still updates happening in the log file which are still monitored by the Agent.


wisdom : sudo
File Edit View Scrollback Bookmarks Settings Help
Defining the communication channel
wisdom : sudo

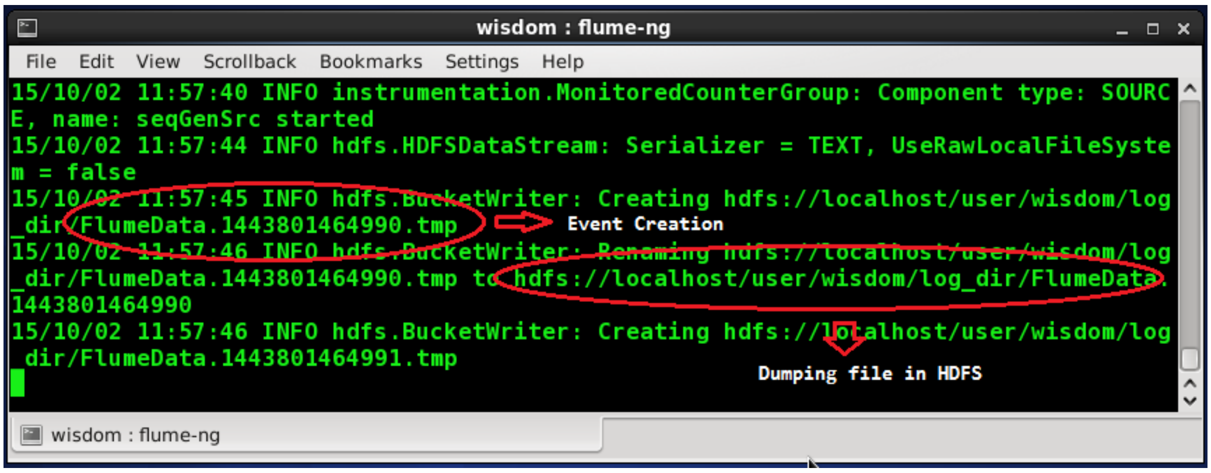
```
agent.channels = memoryChannel
agent.sinks = loggerSink

# For each one of the sources, the type is defined
agent.sources.seqGenSrc.type = exec
agent.sources.seqGenSrc.command = tail -F /home/wisdom/Documents/log.txt

# The channel can be defined as follows.
agent.sources.seqGenSrc.channels = memoryChannel

# Each sink's type must be defined
agent.sinks.loggerSink.type = logger

#Specify the channel the sink should use
agent.sinks.loggerSink.channel = memoryChannel

-- INSERT --
```

Fig 59: Defining the Communication Channel


wisdom : sudo
File Edit View Scrollback Bookmarks Settings Help
Defining the HDFS Destination
wisdom : sudo

```
agent.sinks.loggerSink.type = logger

#Specify the channel the sink should use
agent.sinks.loggerSink.channel = memoryChannel

# Each channel's type is defined.
agent.channels.memoryChannel.type = memory

# Other config values specific to each type of channel(sink or source)
# can be defined as well
# In this case, it specifies the capacity of the memory channel
agent.channels.memoryChannel.capacity = 100

agent.sinks.loggerSink.type = hdfs
agent.sinks.loggerSink.hdfs.path = hdfs://localhost/user/wisdom/logs
agent.sinks.loggerSink.hdfs.fileType = DataStream
-- INSERT --
```

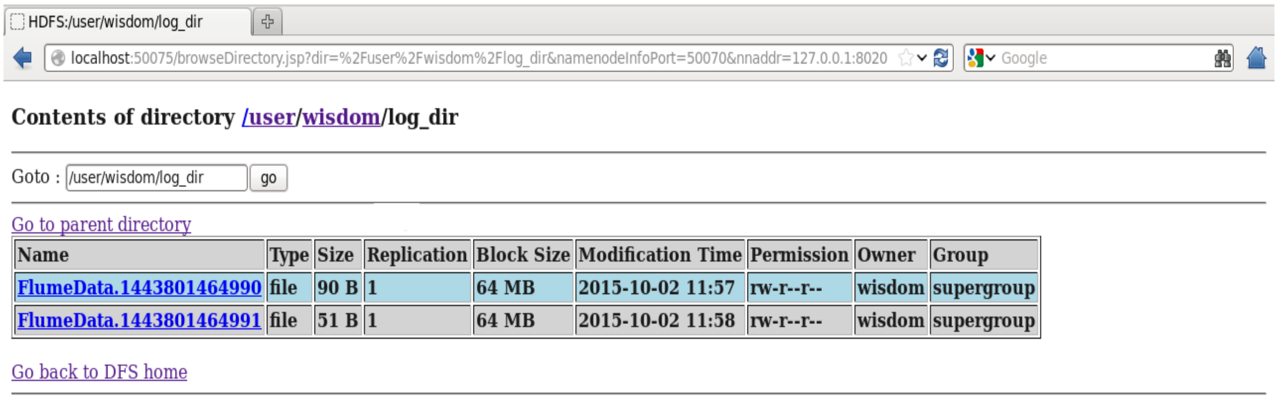
Fig 60: Defining the Destination



```
15/10/02 11:57:40 INFO instrumentation.MonitoredCounterGroup: Component type: SOURC
E, name: seqGenSrc started
15/10/02 11:57:44 INFO hdfs.HDFSDataStream: Serializer = TEXT, UseRawLocalFileSyste
m = false
15/10/02 11:57:45 INFO hdfs.BucketWriter: Creating hdfs://localhost/user/wisdom/log
_dir/FlumeData.1443801464990.tmp
15/10/02 11:57:46 INFO hdfs.BucketWriter: Renaming hdfs://localhost/user/wisdom/log
_dir/FlumeData.1443801464990.tmp to hdfs://localhost/user/wisdom/log_dir/FlumeData.
1443801464990
15/10/02 11:57:46 INFO hdfs.BucketWriter: Creating hdfs://localhost/user/wisdom/log
_dir/FlumeData.1443801464991.tmp
```

Fig 61: FLUME Agent Listening Process

This process is continuous and will carry on till you explicitly terminate the FLUME agent. In this way we can have all the contents of the log file as and when they are generated.



HDFS:/user/wisdom/log_dir
localhost:50075/browseDirectory.jsp?dir=%2Fuser%2Fwisdom%2Flog_dir&namenodeInfoPort=50070&nnaddr=127.0.0.1:8020
Google

Contents of directory /user/wisdom/log_dir

Goto : /user/wisdom/log_dir go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
FlumeData.1443801464990	file	90 B	1	64 MB	2015-10-02 11:57	rw-r--r--	wisdom	supergroup
FlumeData.1443801464991	file	51 B	1	64 MB	2015-10-02 11:58	rw-r--r--	wisdom	supergroup

Go back to DFS home

Fig 62: FLUME data in HDFS

This is how the contents of an ever incrementing file can be dumped inside Hadoop for further analysis. In the next chapter we will have a look at one of the most important tool in the Hadoop Ecosystem called OOZIE.

INTRODUCTION TO OOZIE

I work as a Freelancer developer and Trainer. Sometimes I do get simultaneous assignments with same deadlines in both the streams. In such situations it becomes very difficult to manage everything.

One day what happened was very interesting. I had to deliver a full day training to some client in their office premises and on the same day I had to deliver 3 development assignments of 3 different clients. The development assignments included executing around 10 commands that were designed as a pipeline. Means output of one operation was used as the input of the next operation. So it was not possible for me to initiate all assignments and leave for the training and hope that they get completed by the time I come back. I needed someone to stay there and execute these commands one after the other and in case of errors suspend the process. Writing a script was also not a feasible solution.

So what I did was, I told my younger brother that he needs to help me today and told him that I would be giving him 3 pieces of paper with 10 commands each and all his has to do is write those commands in the terminal in a sequential manner and in case of error intimate me in the evening.

Assignment 1	Assignment 2	Assignment 3
<name>command 1.1</name>	<name>command 2.1</name>	<name>command 3.1</name>
<action>action 1.1</action>	<action>action 2.1</action>	<action>action 3.1</action>
<name>command 1.2</name>	<name>command 2.2</name>	<name>command 3.2</name>
<action>action 1.2</action>	<action>action 2.2</action>	<action>action 3.2</action>
<name>command 1.3</name>	<name>command 2.3</name>	<name>command 3.3</name>
<action>action 1.3</action>	<action>action 2.3</action>	<action>action 3.3</action>
<name>command 1.4</name>	<name>command 2.4</name>	<name>command 3.4</name>
<action>action 1.4</action>	<action>action 2.4</action>	<action>action 3.4</action>

Fig 63: Sample Workflow

Though he was not aware about what actually the actions were or what was the outcome of what he is doing, he still completed all 3 assignments by the time I came back. It was a big help and this solution really worked nice for me.

Was this incident interesting? Now let's relate it to what we have to understand in this chapter.

In the above incident I represent any Hadoop Client that has multiple interdependent jobs to be executed on the Hadoop Cluster. My younger brother is the Hadoop Ecosystem tool called OOZIE. The pieces of paper are xml documents called as OOZIE workflows inside which the name represents the Control Flow Nodes and the action represents the Action Nodes.

WHAT IS OOZIE

OOZIE is a Hadoop Ecosystem tool which works as a Workflow Scheduler. What OOZIE can do is simply schedule your jobs and execute them in a sequential manner. OOZIE is not concerned whether it is a Map reduce job or a Pig job or a Hive job or any other Hadoop Job. It is only concerned with a xml file which called as OOZIE Workflow that defines the actions to be performed by the engine in a sequential manner.

If a PIG command is to be executed, the OOZIE engine will simply submit that to the PIG engine and will make sure that it gets executed and the output and/or error are given back as the result.

Apart from this, OOZIE can also work like a CRON job. You can decide a schedule for your jobs and OOZIE will execute the job whenever the time triggers.

With this we complete the introduction of the Hadoop Ecosystem. In the next chapter we will have a look at how to plan and set up our Hadoop Cluster.

HADOOP CLUSTER PLANNING

Planning deployment of a Hadoop Cluster involves analysing and deciding on a number of crucial aspects like version and distribution, Size of Cluster, Hardware Configurations, Ports and Properties, Job Configurations etc. In this chapter we will have a brief look into these aspects from a Hadoop Administrator's point of view.

CAPACITY PLANNING

The very first decision will remain on how big your Hadoop Cluster should be. This decision will be influenced by the amount of data that your company handles.

FOR EXAMPLE:

Your company collects around 2 TB of data every month. With default replication factor, you would be dumping around 6 TB of data every month in your Hadoop Cluster. Additionally not all storage space is allotted to the HDFS. So every month your company will be needing approximately 8 TB of storage.

Considering you buy servers with storage capacity of 4 TB, then you will need around 2 servers in a month which approximates to a total of around 24 Data Nodes in a year. Additionally your Cluster will have the master servers.

Depending on the company's budget, you can either have the Name Node, Job Tracker and Secondary Name Node on separate servers, which makes a total of 24 + 3 = 27 Nodes or you may opt to have the Name Node and Job Tracker on same servers and the Secondary Name Node on separate servers, which makes a total of 24 + 2 = 26 Nodes.

HARDWARE SELECTION

There are generally 2 types of nodes in a Hadoop Cluster viz. Master Nodes and Slave Nodes.

Master nodes are crucial in the performance of the Hadoop Cluster and are the single point of failures. Name Node going down will leave the entire cluster inaccessible and if the Job Tracker goes down you will have to resubmit all the running ad queued jobs. So make sure that you purchase best in class servers with at least the following specifications

1. QUAD CORE CPU
2. 32 GB RAM
3. RAIDED HARD DISK DRIVES
4. DUAL POWER SUPPLIES
5. DUAL ETHERNET CARDS

Slave Nodes are the actual working Nodes. In order to achieve higher rate of parallel processing, it is suggested to trust in quantity instead of quality. Means if you can either buy 2 best quality servers or buy 5 less efficient servers, then it is always better to go with more number of nodes.

Typically the hardware configurations for a slave Node may fall under the ratio of 1:2:6-8. Means if you have 1 TB of Storage, then you may have atleast 2 cores of CPU and atleast 6 to 8 GB of RAM. The number of simultaneous Map and Reduce tasks is equal to 1.5 times of the number of CPU cores in the slave server. Means if you have around 8 cores of CPU then that machine can perform around 12 simultaneous tasks.

OS SELECTION

Hadoop can be installed on any Linux Operating system *(Windows only if Hortonworks is used)* like RHEL, Fedora, Ubuntu, Centos etc. Normally in a production deployment, RHEL is used on the master servers and Centos is used on the slave nodes. Feel free to choose an OS that your company is comfortable administering.

JAVA SELECTION

Being written in JAVA, this becomes one of the major factors to think upon. Depending on how you are going to write, run and execute the programs, you can choose between JDK and JRE. Hadoop supports both. As

Hadoop is way too complex in itself, it is recommended to choose the most stable version of JAVA, to avoid additional complexity out of the bugs in unstable versions. You can download JAVA from Oracle or Sun's official sites.

HADOOP SELECTION

Selecting the version and distribution of Hadoop depends on a number of factors but what is more important is being able to map your requirements with the available features and investing in only what is required.

HADOOP HISTORY

Deploying the recent most stable version is what any company would like to do. This will surely give a lot of features but initially it is better to select a stable version based on requirement so that you don't increase the complexity with unnecessary implementations.

Hadoop has grown from a fundamental HDFS with Map Reduce to a large Ecosystem over a period of 10 years now. The below tables will help you in understanding the different releases of Hadoop

Version	**Features**
4 September, 2007 : release 0.14.1	Better checksums in HDFS. Introduced C++ API for MapReduce. Eclipse Plugin, including HDFS browsing, job monitoring, etc.
29 October 2007: release 0.15.0	contains the first working version of Hbase
24 February, 2009: release 0.19.1	Issues related to Data Loss were resolved
23 August, 2010: release 0.21.0	Stable version for Production deployments
11 May, 2011: release 0.20.203.0	First version to be successfully deployed on 4,500 machine cluster
5 Sep, 2011: release 0.20.204.0	RPMs and DEBs were introduced for the first time

Fig 64: Generation 1 Releases

Version	Features
11 Nov, 2011: release 0.23.0	Contains HDFS Federation and YARN
10 December, 2011: release 0.22.0	Some new features were added like 1. HBase support with hflush and hsync. 2. New implementation of file append. 3. BackupNode and CheckpointNode. 4. Hierarchical job queues. 5. Job limits per queue/pool. 6. Dynamically stop/start job queues. 7. Andvances in new mapreduce API: Input/Output formats, ChainMapper/Reducer. 8. TaskTracker blacklisting. 9. DistributedCache sharing.
27 December, 2011: release 1.0.0	Provides major features likes: 1. Security 2. HBase (append/hsynch/hflush, and security) 3. Webhdfs (with full support for security) 4. Performance enhanced access to local files for HBase
10 Mar, 2012: Release 1.0.1 available	Provided better compatibility with Ganglia, HBase, and Sqoop
9 October, 2012: Release 2.0.2-alpha	This delivers significant enhancements to HDFS HA. Also it has a significantly more stable version of YARN

Fig 65: Generation 2 Releases

Version	Features
14 February, 2013: Release 2.0.3-alpha	This version introduced improved features like: 1. QJM for HDFS HA for NameNode 2. Multi-resource scheduling (CPU and memory) for YARN 3. YARN ResourceManager Restart 4. Significant stability at scale for YARN (over 30,000 nodes and 14 million)
13 May, 2013: Release 1.2.0	This release delivers over 200 enhancements and bug-fixes. Major enhancements include: 1. Web services for JobTracker 2. WebHDFS enhancements 3. More robust Namenode in case of edit log corruption 4. Add NodeGroups level to NetworkTopology
25 August, 2013: Release 2.1.0-beta	This relaese has significant changes like: 1. Improved HDFS Snapshots 2. Support for running Hadoop on Microsoft Windows 3. YARN API stabilization 4. Binary Compatibility for MapReduce applications built on hadoop-1.x 5. Substantial amount of integration testing with rest of projects in the ecosystem
15 October, 2013: Release 2.2.0	Apache Hadoop 2.2.0 is the GA release of Apache Hadoop 2.x. This release has a number of significant highlights compared to Hadoop 1.x: 1. YARN - A general purpose resource management system for Hadoop to allow MapReduce and other other data processing frameworks and services 2. High Availability for HDFS 3. HDFS Federation 4. HDFS Snapshots 5. NFSv3 access to data in HDFS 6. Support for running Hadoop on Microsoft Windows 7. Binary Compatibility for MapReduce applications built on hadoop-1.x 8. Substantial amount of integration testing with rest of projects in the ecosystem

Fig 66: Bug Fixing Releases

20 February, 2014: Release 2.3.0	Support for Heterogeneous Storage hierarchy in HDFS. In-memory cache for HDFS data with centralized administration and management. Simplified distribution of MapReduce binaries via HDFS in YARN Distributed Cache.
18 November, 2014: Release 2.6.0	Apache Hadoop 2.6.0 contains a number of significant enhancements such as: 1. Hadoop Common a. Key management server (beta) b. Credential provider (beta) 2. Hadoop HDFS a. Support for Archival Storage b. Transparent data at rest encryption (beta) c. Operating secure DataNode without requiring root access d. Hot swap drive: support add/remove data node volumes without restarting data node (beta) 3. Hadoop YARN a. Support for long running services in YARN i. Service Registry for applications b. Support for rolling upgrades i. Work-preserving restarts of ResourceManager ii. Container-preserving restart of NodeManager c. Support node labels during scheduling d. Support for time-based resource reservations in Capacity Scheduler (beta)
hadoop-3.0	Major changes include 1. HADOOP a. Move to JDK8+ b. Classpath isolation on by default c. Shell script rewrite 2. HDFS a. Removal of hftp in favor of webhdfs b. Support for Erasure Codes in HDFS 3. YARN MAPREDUCE a. Derive heap size or mapreduce.*.memory.mb automatically

Fig 67: Recent Editions

EVOLUTION OF HADOOP ECOSYSTEM

Developments did took place not only within Hadoop but also around it.

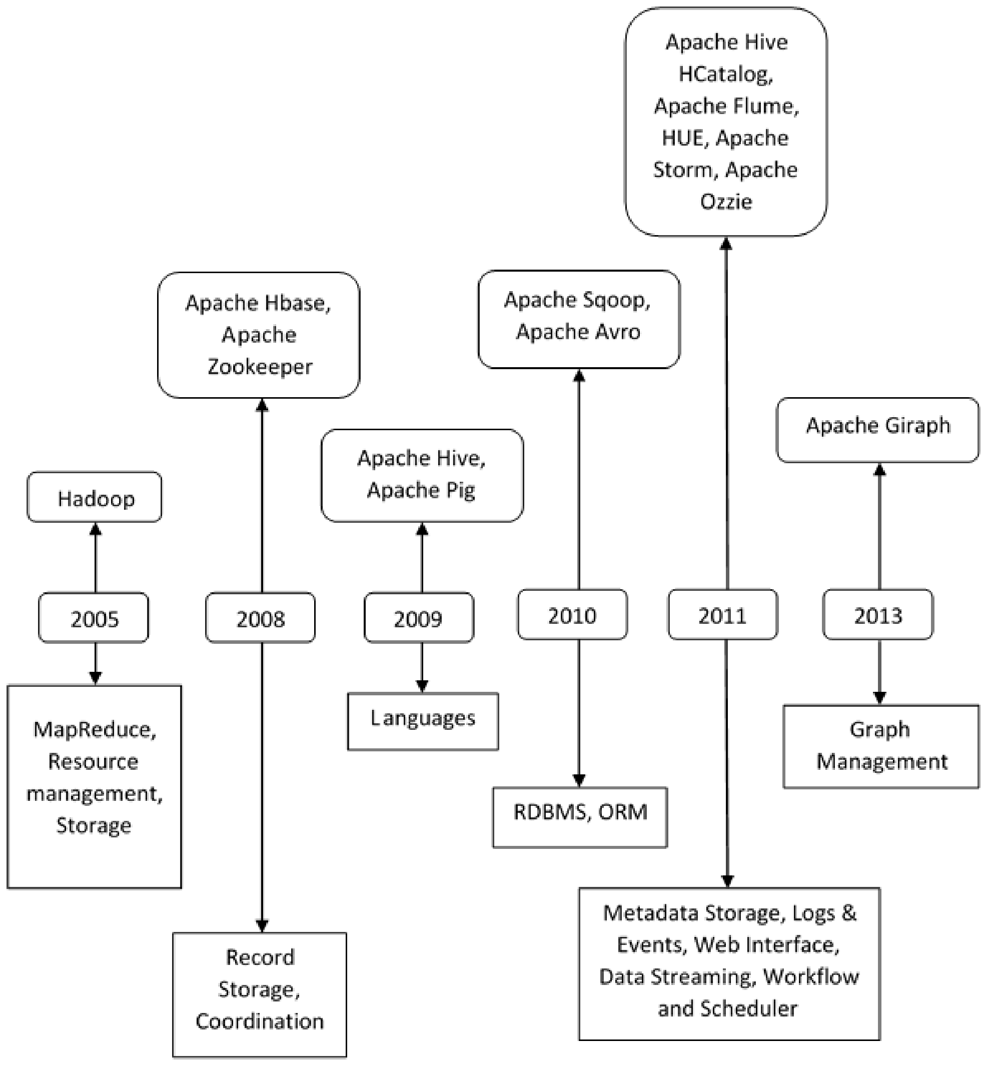

Apache Hive HCatalog, Apache Flume, HUE, Apache Storm, Apache Ozzie
Apache Hbase, Apache Zookeeper
Apache Sqoop, Apache Avro
Apache Giraph
Apache Hive, Apache Pig
Hadoop
2005
2008
2009
2010
2011
2013
MapReduce, Resource management, Storage
Languages
Graph Management
RDBMS, ORM
Record Storage, Coordination
Metadata Storage, Logs & Events, Web Interface, Data Streaming, Workflow and Scheduler

Fig 68: Hadoop Evolution

All these Ecosystem tools were to be installed and managed individually. Because of this, companies working on large applications with Hadoop started to face difficulties giving rise to Hadoop Distributions.

HADOOP DISTRIBUTIONS

Hadoop Distributions include organisations that provide packaged Hadoop Ecosystem, support etc. These organisations have created a platform where most of the popular Hadoop Ecosystem tools are integrated together along with the core functionalities to provide a single interface for installing and managing the vast range of Ecosystem tools. The distributions have not only made deployment of large applications around Hadoop easy but also provide immediate support over bugs and problems to companies that use their platforms.

Out of the many distributions available Cloudera, Hortonworks and MapR are the most popular ones. All these distributions have their own characteristics in the number and type of tools bundled in the package, the support, the new releases etc. Based on the requirements of the company, these characteristics influence the decision of selecting one of these distributions.

CLOUDERA'S DISTRIBUTION OF HADOOP

Cloudera Inc. is an American-based software company that provides Apache Hadoop-based software, support and services, and training to business customers.

Cloudera's open-source Apache Hadoop distribution, CDH *(Cloudera Distribution Including Apache Hadoop)*, targets enterprise-class deployments of that technology. Cloudera says that more than 50% of its engineering output is donated upstream to the various Apache-licensed open source projects *(Apache Hive, Apache Avro, Apache Hbase, and so on)* that combine to form the Hadoop platform. Cloudera is also a sponsor of the Apache Software Foundation.

Cloudera offers software, services and support in three different bundles:

1. Cloudera Enterprise includes CDH and an annual subscription license *(per node)* to Cloudera Manager and technical support. It comes in three editions: Basic, Flex, and Data Hub.
2. Cloudera Express includes CDH and a version of Cloudera Manager lacking enterprise features such as rolling upgrades and backup/disaster recovery.
3. CDH may be downloaded from Cloudera's website at no charge, but with no technical support nor Cloudera Manager.

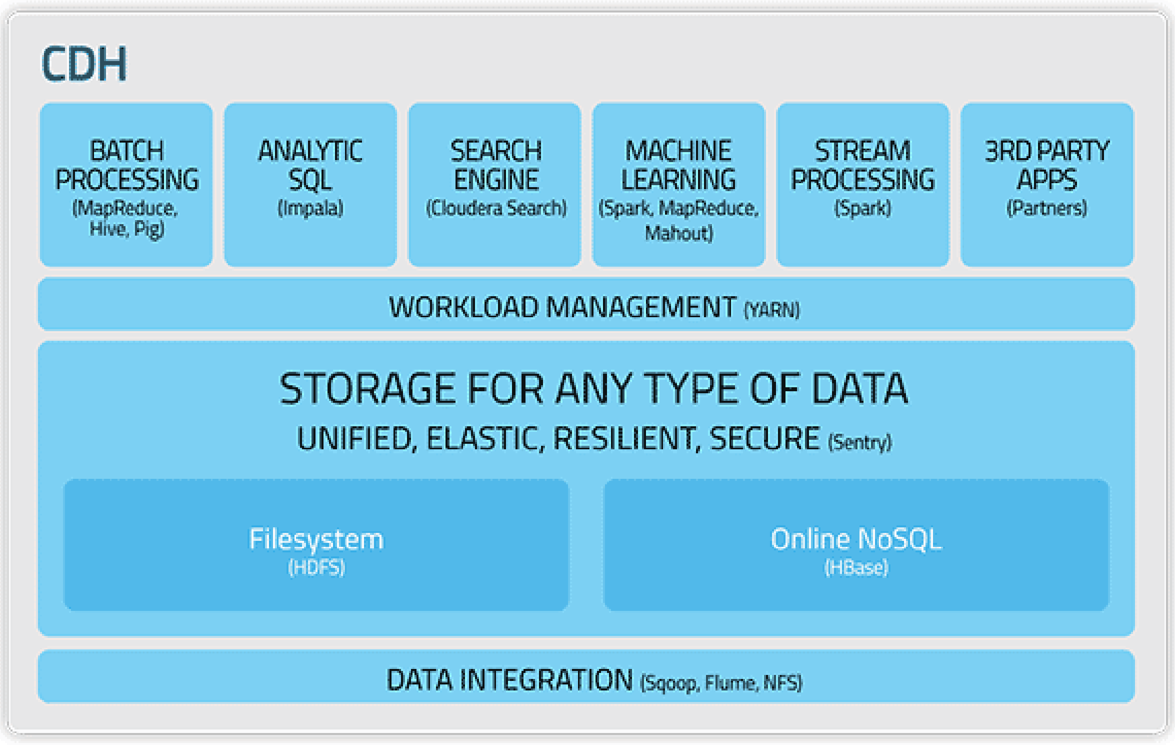

CDH
BATCH PROCESSING (MapReduce, Hive, Pig)
ANALYTIC SQL (Impala)
SEARCH ENGINE (Cloudera Search)
MACHINE LEARNING (Spark, MapReduce, Mahout)
STREAM PROCESSING (Spark)
3RD PARTY APPS (Partners)
WORKLOAD MANAGEMENT (YARN)
STORAGE FOR ANY TYPE OF DATA
UNIFIED, ELASTIC, RESILIENT, SECURE (Sentry)
Filesystem (HDFS)
Online NoSQL (HBase)
DATA INTEGRATION (Sqoop, Flume, NFS)

Fig 69: Cloudera Distribution of Hadoop

HORTONWORKS DATA PLATFORM

Hortonworks is a business computer software company based in Santa Clara, California. The company focuses on the development and support of Apache Hadoop, a framework that allows for the distributed processing of large data sets across clusters of computers.

Hortonworks employs contributors to the open source software project Apache Hadoop. The company was named after Horton the Elephant of the Horton Hears a Who! Book.

Hortonworks is a sponsor of the Apache Software Foundation.

Hortonworks' product named Hortonworks Data Platform (HDP) includes Apache Hadoop and is used for storing, processing, and analysing large volumes of data. The platform is designed to deal with data from many sources and formats. The platform includes various Apache Hadoop projects including the Hadoop Distributed File System, MapReduce, Pig, Hive, Hbase and Zookeeper and additional components.

In October 2011 Hortonworks announced Microsoft would collaborate on a Hadoop distribution for Microsoft Azure and Windows Server. On February 25, 2013, Hortonworks announced availability of a beta version of the Hortonworks Data Platform for Windows.

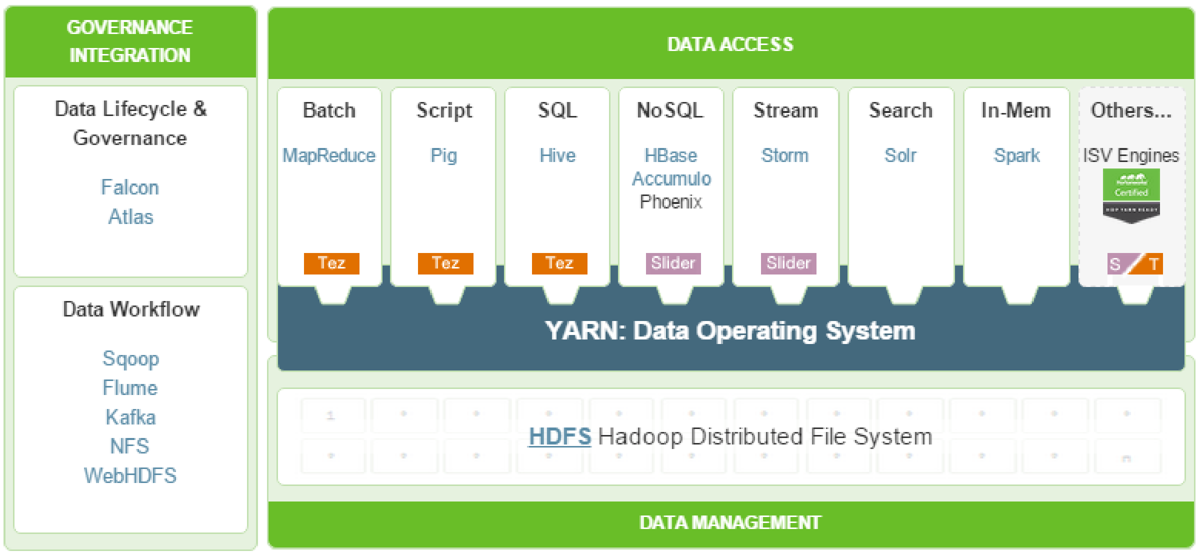


Fig 70: HDP Governance & Data

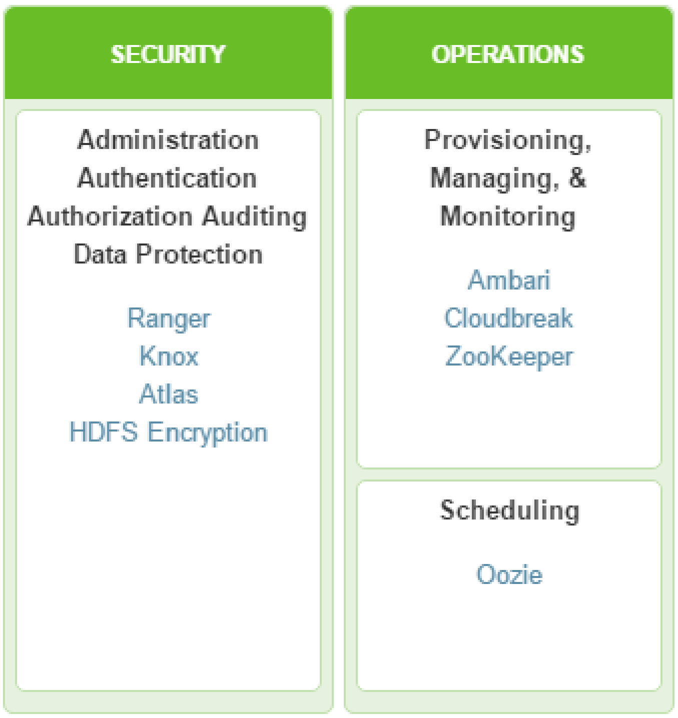


Fig 71: HDP Security & Operations

MAPR

MapR is a San Jose, California-based enterprise software company that develops and sells Apache Hadoop-derived software. The company contributes to Apache Hadoop projects like Hbase, Pig (programming language), Apache Hive, and Apache ZooKeeper. MapR's Apache Hadoop distribution claims to provide full data protection, no single points of failure, improved performance, and dramatic ease of use advantages. MapR entered a technology licensing agreement with EMC Corporation on 25 May 2011, supporting an EMC-specific distribution of Apache Hadoop. MapR was selected by Amazon to provide an upgraded version of Amazon's Elastic Map Reduce *(EMR)* service. MapR has also been selected by Google as a technology partner. MapR was able to break the minute sort speed record on Google's compute platform.

MapR provides three versions of their product known as M3, M5 and M7. M3 is a free version of the M5 product with degraded availability features. M7 is like M5, but adds a purpose built rewrite of Hbase that implements the Hbase API directly in the file-system layer.

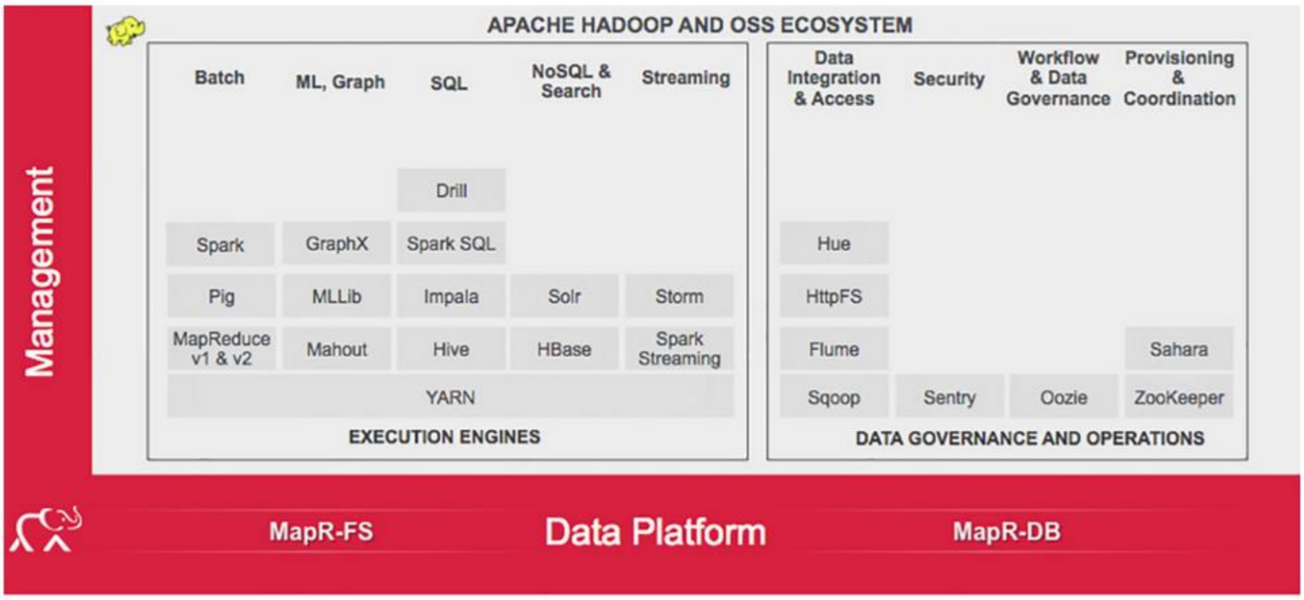


Fig 72: MapR Hadoop Distribution

OTHER HADOOP DISTRIBUTIONS

1. DATAMEER

Datameer Analytics Solution (DAS) is a Hadoop-based solution for big data analytics that includes data source integration, storage, an analytics engine and visualization.

2. IBM

IBM InfoSphere BigInsights brings the power of Apache Hadoop to the enterprise. BigInsights Enterprise Edition builds on Apache Hadoop with capabilities to withstand the demands of an enterprise including:

1. Advanced Text Analytics
2. Performance Optimizations
3. Workload Management & Scheduling
4. Professional-Grade Visualization & Developer Tooling
5. Enterprise Integration & Security

3. AMAZON WEB SERVICES

Amazon offers a version of Apache Hadoop on their EC2 infrastructure, sold as Amazon Elastic MapReduce.

4. APACHE BIGTOP

Apache Bigtop is a project for the development of packaging and tests of the Apache Hadoop ecosystem. This includes testing at various levels (packaging, platform, runtime, upgrade, etc...) developed by a community with a focus on the system as a whole, rather than individual projects.

5. CLOUDSPACE

Cloudspace is a web technology consulting company, since 1996. Cloudspace uses Apache Hadoop to scale client and internal projects on Amazon's EC2 and bare metal architectures.

6. CASCADING

Cascading is a popular feature-rich API for defining and executing complex and fault tolerant data processing workflows on a Apache Hadoop cluster. Cascading 2.0 is Apache-licensed.

7. DATASTAX

DataStax provides a product which fully integrates Apache Hadoop with Apache Cassandra and Apache Solr in its DataStax Enterprise platform. DataStaxEnterprise is completely free to use for development environments with no restrictions.

8. PENTAHO – OPEN SOURCE BUSINESS INTELLIGENCE

Pentaho provides a complete, end-to-end open-source BI alternative to proprietary offerings like Oracle, SAP and IBM.

9. VMWARE

1. Initiate Open Source project and product to enable easily and efficiently deploy and use Hadoop on virtual infrastructure.
2. HVE - Hadoop Virtual Extensions to make Hadoop virtualization-aware
3. Serengeti - enable the rapid deployment of an Apache Hadoop cluster

HADOOP INSTALLATION MODES

Hadoop can be installed and configured in 2 modes

1. PSEUDO DISTRIBUTED MODE

In this mode all the Hadoop services viz. masters as well as slaves are configured and started in one single machine. This is also called as a Single Node Hadoop Cluster. This mode is useful for testing and learning purpose but won't help you in dealing with Big Data.

2. DISTRIBUTED MODE

This is the mode for which Hadoop was actually built. This is also called as a Multi Node Cluster. This Shared Nothing architecture makes sure that you master and slaves are installed and configured on different servers. Normally Name Node and Job Tracker are installed on one server and Data Node and Task Trackers are installed on the slave servers.

IMPORTANT CONFIGURATION FILES

In the Hadoop installation, the most important configuration files are

1. CORE-SITE.XML

In this file you will specify the changes that deal with the cluster information. For example the details of the master node will be specified in this file. The configuration of client side ecosystem tools can also be specified in this file.

2. HDFS-SITE.XML

In this file you will specify configurations related to HDFS data storage and retrieval. Configurations like the Replication Factor, Block Size etc. can be modified from this file.

3. MAPRED-SITE.XML

In this file you can play with the default configurations for executing the Map Reduce programs. Like if you want to have 4 Mappers for all your operations then you can specify it from this file.

The settings in these files are called as the configurations on the cluster and has the least priority if the same configurations are specified from elsewhere. There are 3 ways in which you can specify or modify the configurations.

1. From the Program (highest priority)
2. From the Client (second highest priority)
3. From the Cluster (lowest priority)

After analysing all these aspects in the most appropriate way, you can possibly end up setting a cluster with a high cost/performance ratio.

www.wisdomsprouts.com

WISDOM SPROUTS

We Foster Knowledge Here