CHAPTER 1 INTRODUCTION

An Overview

- Big Data?
- The V's of Big Data: Adding Value.
- Big Data and its evolution
- Solution to Big Data- Hadoop
- Brief History of Hadoop
- Why Hadoop
- Hadoop Architectural Overview
- Why is Big Data important?

1.1 - Big Data?

In order to understand 'Big Data', we first need to know what 'data' is. Oxford dictionary defines 'data' as –

"The quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media. "Big data is a blanket term for the non-traditional strategies and technologies needed to gather, organize, process, and gather insights from large datasets. While the problem of working with data that exceeds the computing power or storage of a single computer is not new, the pervasiveness, scale, and value of this type of computing has greatly expanded in recent years. An exact definition of "big data" is difficult to nail down because projects, vendors, practitioners, and business professionals use it quite differently. With that in mind, generally speaking, big data is:

- **large datasets**
- the category of computing strategies and technologies that are used to handle large datasets

In this context, "large dataset" means a dataset too large to reasonably process or store with traditional tooling or on a single computer. This means that the common scale of big datasets is constantly shifting and may vary significantly from organization to organization.

Big data' could be found in three forms:

- **Structured**
- **Unstructured**
- **Semi-structured**

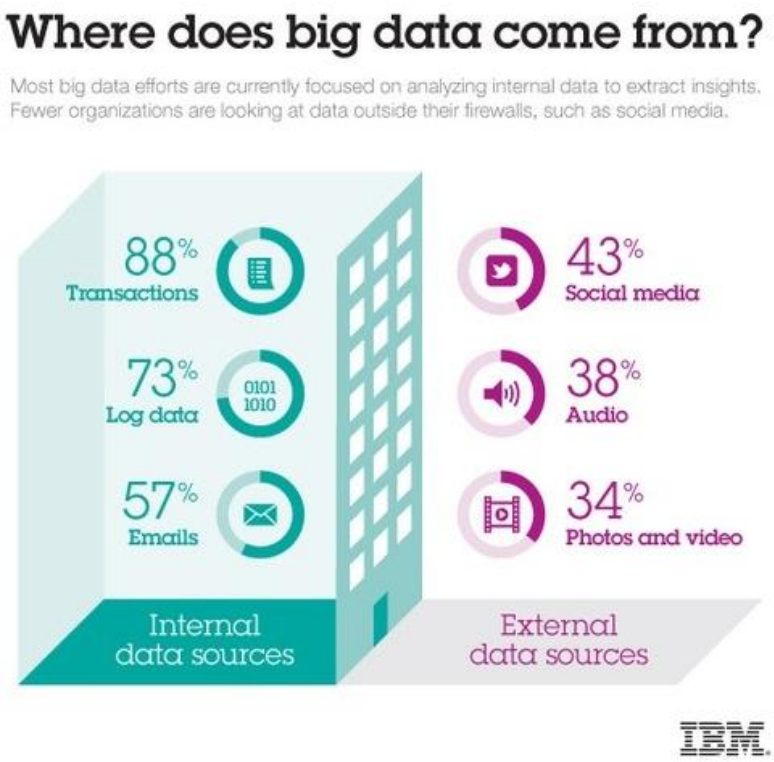


Figure 1.1 – Sources of Big Data _credit:IBM

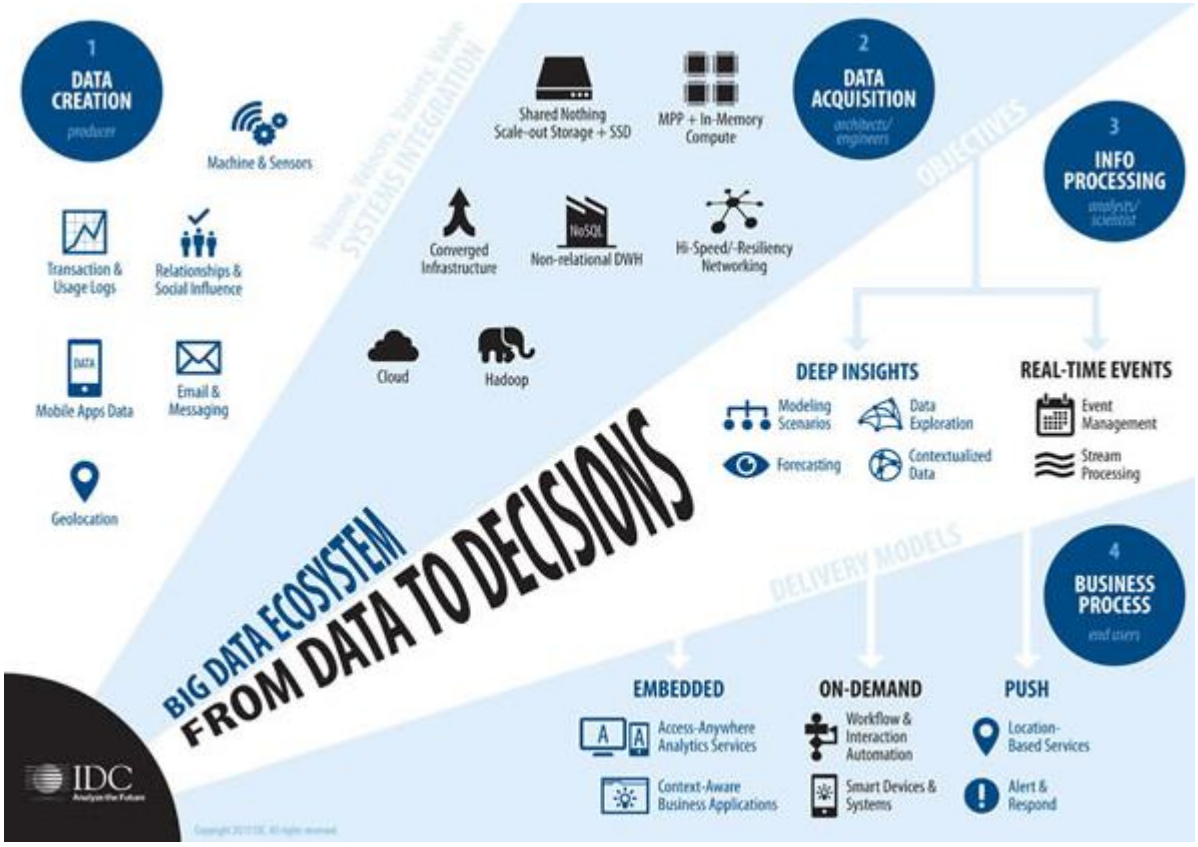


Figure 1.2 Big data ecosystem- from data to decisions.

Today, and certainly here, we look at the business, intelligence, decision and value/opportunity perspective. From volume to value (what data do we need to create which benefit) and from chaos to mining and meaning, putting the emphasis on data analytics, insights and action**.** The world's technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s; as of 2012, every day 2.5 exabytes (2.5×1018) of data are generated.[1] By 2025, IDC predicts there will be 163 zettabytes of data. One question for large enterprises is determining who should own big-data initiatives that affect the entire organization.

1.2 – The V's of Big Data: Adding Value.

Value is an essential 'V' that should be added to the so-called three big data 'V's'.

IBM added a fourth one to that the list of the classic three V's as data is increasing exponentially in an era of ongoing digitization.: [2]

- **Volume** - The sheer volume of data and information that gets created whereby we mainly talk infrastructure, processing and management of big data, be it in a selective way.
- **Velocity** - Velocity is about where analysis, action and also fast capture, processing and understanding happen and where we also look at the speed and mechanisms at which large amounts of data can be processed for increasingly near-time or real-time outcomes, often leading to the need of fast data.
- **Variety** - On top of the data produced in a broad digital context, regardless of business function, societal area or systems, there is a huge increase in data created on more specific levels. Variety is about the many types, being structured, unstructured and everything in between.
- **Veracity** - Veracity has everything to do with accuracy which from a decision and intelligence viewpoint becomes certainty and the degree in which we can trust upon the data to do what we need/want to do.
- **Value** - As said we add value to that as it's about the goal, the outcome, the prioritization and the overall value and relevance created in Big Data applications, whereby the value lies in the eye of the beholder and the stakeholder and never or rarely in the volume dimension. Welcome to Big Data in Action.

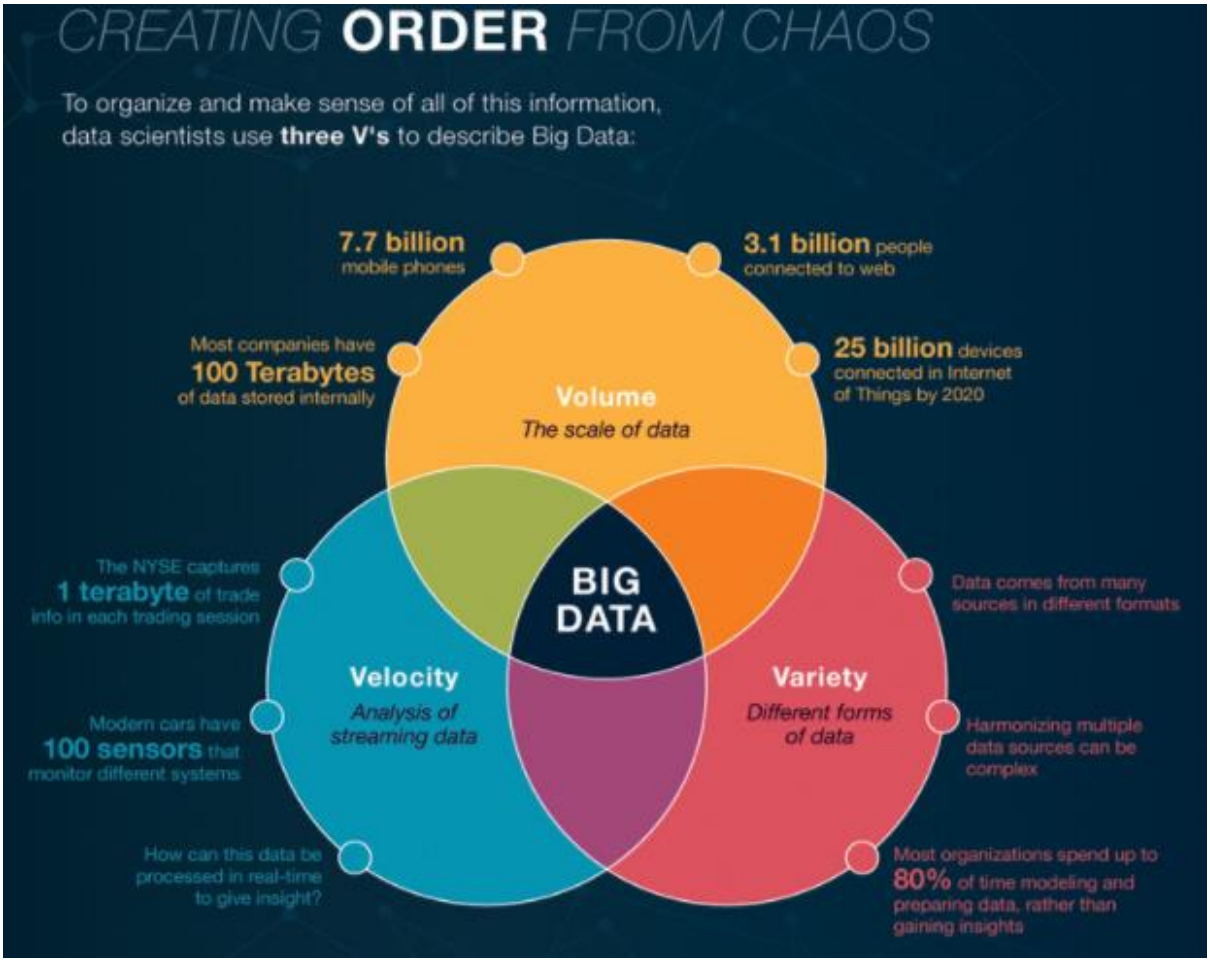


Figure – 1.3 V's of Big Data

Developed economies increasingly use data-intensive technologies. There are 4.6 billion mobile-phone subscriptions worldwide, and between 1 billion and 2 billion people accessing the internet.[3] Between 1990 and 2005, more than 1 billion people worldwide entered the middle class, which means more people became more literate, which in turn lead to information growth. The world's effective capacity to exchange information through telecommunication networks was 281 petabytes in 1986, 471 petabytes in 1993, 2.2 exabytes in 2000, 65 exabytes in 2007 and predictions put the amount of internet traffic at 667 exabytes annually by 2014. According to one estimate, one-third of the globally stored information is in the form of alphanumeric text and still image data, which is the format most useful for most big data applications.

This flood of data is coming from many sources. Consider the following:

• The New York Stock Exchange generates about 4−5 terabytes of data per day.

• Facebook hosts more than 240 billion photos, growing at 7 petabytes per month.

• Ancestry.com, the genealogy site, stores around 10 petabytes of data.

• The Internet Archive stores around 18.5 petabytes of data.

• The Large Hadron Collider near Geneva, Switzerland, produces about 30 petabytes of data per year

1.3 – Big Data and its evolution

1.3.1 **Smart data**

beyond the volume and towards the reality: Big data is…big. With increasing volumes of mainly unstructured data comes a challenge of noise within the sheer volume aspect.

In order to achieve business outcomes and practical outcomes to improve business, serve customer betters, enhance marketing optimization or respond to any kind of business challenge that can be improved using data, we need smart data whereby the focus shifts from volume to value.

1.3.2 **Fast data**: speed and agility for responsiveness

In order to react and pro-act, speed is of the utmost importance.However, how do you move from the – mainly unstructured – data avalanche that big data really is to the speed you need in a real-time economy? Fast data is one of the answers in times when customer-adaptiveness is key to maintain relevance.

1.3.3 **Solving the Big Data challenge with artificial intelligence**

Roland Simonis explains how artificial intelligence is used for Intelligent Document Recognition and the unstructured information and big data challenges.

1.3.4 **Unstructured data: adding meaning and value**

The largest and fastest growing form of information in the Big Data landscape is what we call unstructured data or unstructured information. Coming from a variety of sources it adds to the vast and increasingly diverse data and information universe.

To turn the vast opportunities in unstructured data and information (ranging from text files and social data to the body text of an email), meaning and context needs to be derived. This is what cognitive computing enables: seeing patterns, extracting meaning and adding a “why” to the “how” of Big Data.

1.4 – Solution to Big Data- Hadoop

Figure 1.4 - Hadoop logo (The Yellow Elephant)

Apache Hadoop is an open-source software framework used for distributed storage and processing of data set of big data using the MapReduce programming model. It consists of computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework. [4]

It provides scaling in:

Storage

Performance

I/O Bandwidth

- What makes Hadoop special? No high end or expensive systems are required. Built on commodity hardware.

Can run on your machine!

- Can run on Linux, Mac OS/X, Windows, Solaris.
- No discrimination as its written in java.
- Fault tolerant system!
- Execution of the job continues even of nodes are failing.
- It accepts failure as part of system.
  - Highly reliable and efficient storage system.

  -In built intelligence to speed up the applications.

- Speculative execution.
- Fit for lot of applications:
- Web log processing.
- Page Indexing, page ranking.
- Complex event processing.

Acording to IDC the “Worldwide Big Data and Business Analytics Market” or BDA, so analytics alone, is poised to grow from $130.1 billion this year to over $203 billion in 2020 (forecastpublished on October 3rd, 2016), among others driven by a shift towards a data-driven mindset. “The availability of data, a new generation of technology, and a cultural shift toward data-driven decision making continue to drive demand for big data and analytics technology and services”.

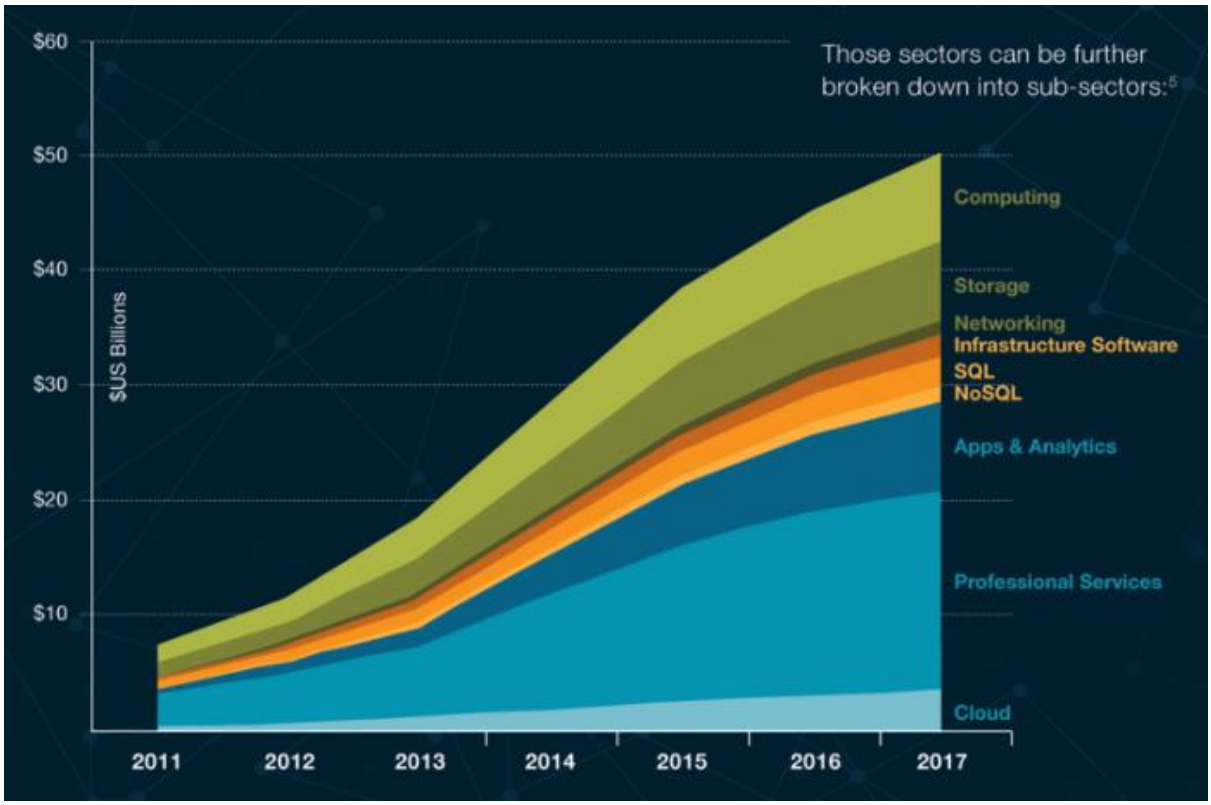


Figure 1.5 - Forcast

1.5 - Brief History of Hadoop

According to its co-founders, Doug Cutting and Mike Cafarella, the genesis of Hadoop was the "Google File System" paper that was published in October 2003.[5] This paper spawned another one from Google – "MapReduce: Simplified Data Processing on Large Clusters". Development started on the Apache Nutch project, but

was moved to the new Hadoop subproject in January 2006. Doug Cutting, who was working at Yahoo! at the time, named it after his son's toy elephant. The initial code that was factored out of Nutch consisted of about 5,000 lines of code for HDFS and about 6,000 lines of code for MapReduce.

The first committer to add to the Hadoop project was Owen O'Malley (in March 2006);[5] Hadoop 0.1.0 was released in April 2006.[5] It continues to evolve through the many contributions that are being made to the project.

Oct 2003 – Google GFS paper published.

Dec 2004 – MapReduce: Simplified Data processing

Jul 2005 – Nutch uses MapReduce

Feb 2006 – Starts as a Lucene subproject

Apr 2006 – Hadoop 0.1.0 released

Apr 2007 – Yahoo! On 1000-node cluster

Jan 2008 – An Apache Top level Project

Jan 2008 – YARN JIRA opened

Jul 2008 – A 4000-node cluster

Mar 2009 – Yahoo! runs 17 clusters with 24,000 machines

Apr 2009 – Hadoop sorts a Petabyte in 17 hours

Jan 2011 – Facebook, LinkedIn, eBay and IBM collectively contribute 200,000 lines of code

Nov 2012 – Apache Hadoop 1.0 available

Mar 2013 – YARN deployed in production at Yahoo

Oct 2013 – Apache Hadoop 2.3 available

Jun 2014 – San Jose Hadoop Summit (3,200 attendees)

Jun 2015 – Apache Hadoop 2.7 available

Mar 2017 – Apache Hadoop 2.8 available

1.6 - Why Hadoop?

Over the last 10 years or so, large web companies such as Google, Yahoo!, Amazon and Facebook have successfully applied large scale machine learning algorithms over big data sets, creating innovative data products such as online advertising systems and recommendation engines.

Apache Hadoop is quickly becoming a central store for big data in the enterprise, and thus is a natural platform with which enterprise IT can now apply data science to a variety of business problems such as product recommendation, fraud detection, and sentiment analysis.

REASON 1: DATA EXPLORATION WITH FULL DATASETS:

Data scientists love their working environment. Whether using R, SAS, Mat lab or Python, they always need a laptop with lots of memory to analyze data and build models. In the world of big data, laptop memory is never enough, and sometimes not even close.A common approach is to use a sample of the large dataset, a large a sample as can fit in memory. With Hadoop, you can now run many exploratory data analysis tasks on full datasets, without sampling. Just write a map-reduce job, PIG or HIVE script, launch it directly on Hadoop over the full dataset, and get the results right back to your laptop.

REASON 2: MINING LARGER DATASETS:

In many cases, machine-learning algorithms achieve better results when they have more data to learn from, particularly for techniques such as clustering, outlier detection and product recommenders.

Historically, large datasets were not available or too expensive to acquire and store, and so machine-learning practitioners had to find innovative ways to improve models with rather limited datasets. With Hadoop as a platform that provides linearly scalable storage and processing power, you can now store ALL of the data in RAW format, and use the full dataset to build better, more accurate models.

REASON 3: LARGE SCALE PRE-PROCESSING OF RAW DATA:

As many data scientists will tell you, 80% of data science work is typically with data acquisition, transformation, cleanup and feature extraction. This “pre-processing” step

transforms the raw data into a format consumable by the machine-learning algorithm, typically in a form of a feature matrix.

Hadoop is an ideal platform for implementing this sort of pre-processing efficiently and in a distributed manner over large datasets, using map-reduce or tools like PIG, HIVE, and scripting languages like Python. For example, if your application involves text processing, it is often needed to represent data in word-vector format using TFIDF, which involves counting word frequencies over large corpus of documents, ideal for a batch map-reduce job.

Similarly, if your application requires joining large tables with billions of rows to create feature vectors for each data object, HIVE or PIG are very useful and efficient for this task.

REASON 4: DATA AGILITY:

It is often mentioned that Hadoop is “schema on read”, as opposed to most traditional RDBMS systems which require a strict schema definition before any data can be ingeted into them.

“Schema on read” creates “data agility”: when a new data field is needed, one is not required to go through a lengthy project of schema redesign and database migration in production, which can last months. The positive impact ripples through an organization and very quickly everyone wants to use Hadoop for their project, to achieve the same level of agility, and gain competitive advantage for their business and product line.

1.7 - Hadoop Architectural Overview

Hadoop architecture has three core components:

- Hadoop Distributed File System (**HDFS**)
- MapReduce
- Yet Another Resource Negotiator (**YARN**)

HDFS architecture

The Hadoop Distributed File System (HDFS) is the underlying file system of a Hadoop cluster. It provides scalable, fault-tolerant, rack-aware data storage designed

to be deployed on commodity hardware. Several attributes set HDFS apart from other distributed file systems. Among them, some of the key differentiators are that HDFS are:

- designed with hardware failure in mind
- built for large datasets, with a default block size of 128 MB
- optimized for sequential operations
- rack-aware
- cross-platform and supports heterogeneous clusters

Each block is duplicated twice (for a total of three copies), with the two replicas stored on two nodes in a rack somewhere else in the cluster. If a copy is lost (because of machine failure, for example), HDFS will automatically re-replicate it elsewhere in the cluster, ensuring that the threefold replication factor is maintained.

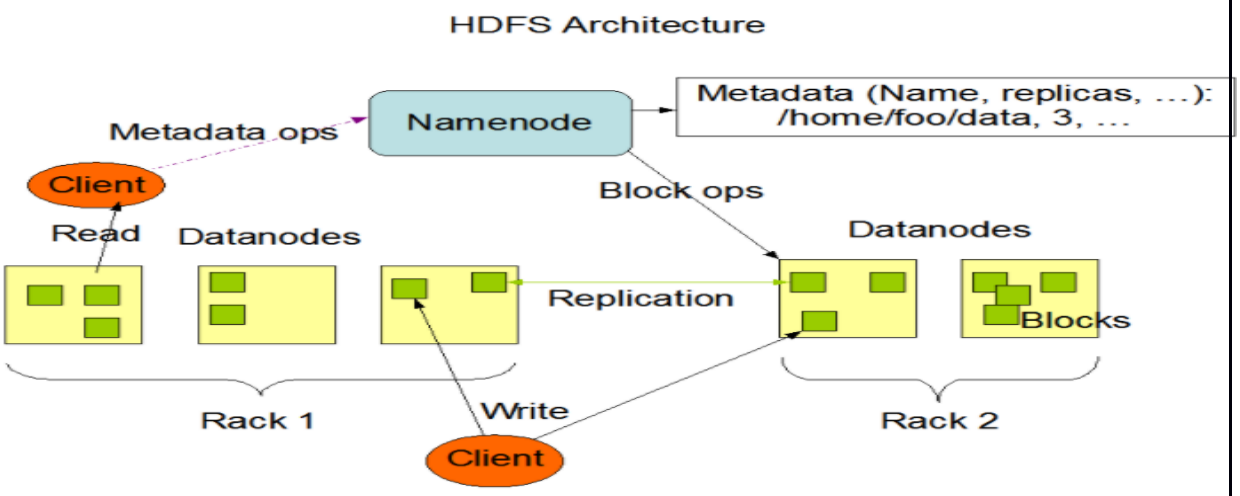


Figure 1.6 HDFS Architecture

- NameNode is single point of contact.
  - Consists of meta information of files stored in HDFS.
  - If it fails, HDFS is inaccessible.
  - HDFS master.
- DataNode consists of actual data.
  - Stored in blocks.
  - Blocks are stored in local file system.
  - HDFS slave

Secondary NameNode does not act as NameNode rather periodically merges the NameNode image with the editlog to prevent editlog from becoming too large.

- MapReduce Overview
  - MapReduce is a framework tailor-made for processing large datasets in a distributed fashion across multiple machines. The core of a MapReduce job can be, err, reduced to three operations: map an input data set into a collection of pairs, shuffle the resulting data (transfer data to the reducers), then reduce over all pairs with the same key.

The top-level unit of work in MapReduce is a job. Each job is composed of one or more map or reduce tasks.

The canonical example of a MapReduce job is counting word frequencies in a body of text. The image below illustrates such an example:

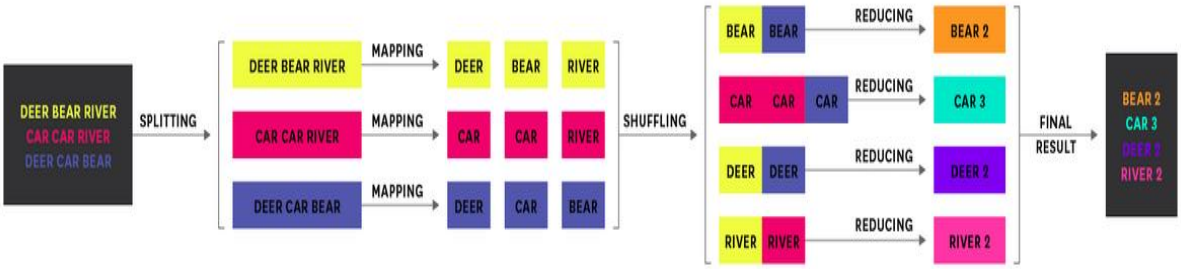


Figure 1.7 Word Counting using MapReduce

In earlier versions of Hadoop (pre-2.0), MapReduce took care of its own resource allocation and job scheduling as well as the actual computation.

Newer versions of Hadoop (2.0+) decouple the scheduling from the computation with YARN, which handles the allocation of computational resources for MapReduce jobs. This allows other processing frameworks to share the cluster without resource contention.

YARN Overview

YARN (Yet Another Resource Negotiator) is the framework responsible for assigning computational resources for application execution.

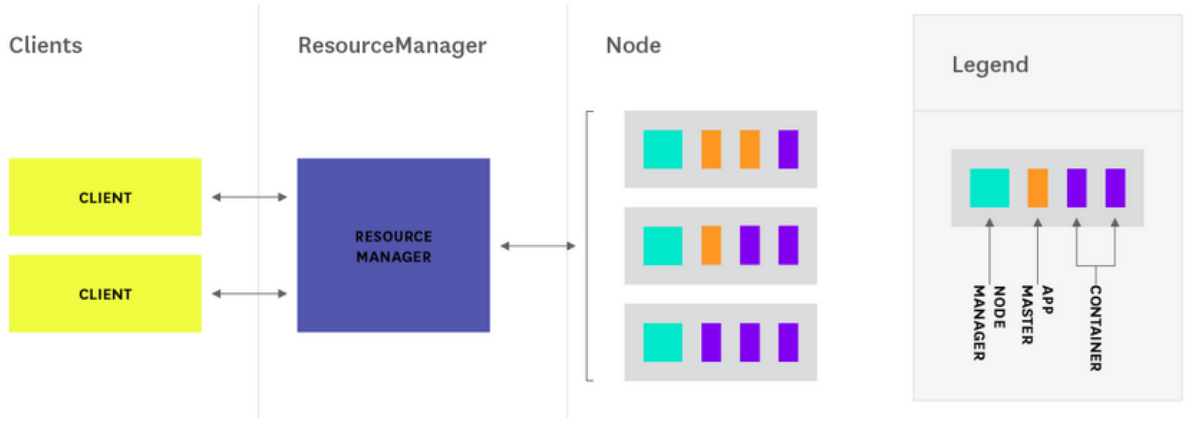


Figure 1.8 YARN interface

YARN consists of three core components:

- ResourceManager (one per cluster)
- ApplicationMaster (one per application)
- NodeManagers (one per node)

The ResourceManager is the rack-aware master node in YARN. It is responsible for taking inventory of available resources and runs several critical services, the most important of which is the Scheduler.

The Scheduler component of the YARN ResourceManager allocates resources to running applications.

Each application running on Hadoop has its own dedicated ApplicationMaster instance. This instance lives in its own, separate container on one of the nodes in the cluster. Each application's ApplicationMaster periodically sends heartbeat messages to the ResourceManager, as well as requests for additional resources, if needed.

A Resource Container (RC) represents a collection of physical resources. It is an abstraction used to bundle resources into distinct, allocatable units.

The NodeManager is a per-node agent tasked with overseeing containers throughout their lifecycles, monitoring container resource usage, and periodically communicating with the ResourceManager.

1.8 – How Big data will change the world?

It is said that the total amount of data in the world doubles every 18 months. Every day, we come up with new ways of collecting information about the habits and patterns that make each of us unique. While it may be fashionable to worry about how

this data could be used to invade our privacy, but the benefit from the data cannot be undermined. Here are the few benefits:

- Walmart example:

Figure 1.9 – Walmart uses big data.

Figure 1.9 – Big data against crime.

- Medicine – Imagine if every time you complained about certain symptoms, your doctor could see the case history on every single patient IN THE ENTIRE WORLD who had experienced those same symptoms. What if your doctor could know exactly how every one of those patients had responded to various treatments and could track your progress as compared to all the rest? Elections - If Big Data analytics can impact our health, our safety, our cities, and what we buy, why not let Big Data choose our leaders as well.In the 2012 presidential election, the Obama Campaign made use of voter models on a scale never before seen. They were able to identify specific voters who would make a difference in the election and target messages to those voters. Therefore it should be remembered the data is nothing if we cannot ask interesting questions form it and thus we can add value to it. What one can unravel depends purely on its imagination and innovation.[6]

CHAPTER 2 SINGLE NODE (PSEUDO MODE) AND MULTI-NODE HADOOP SETUP

An Overview

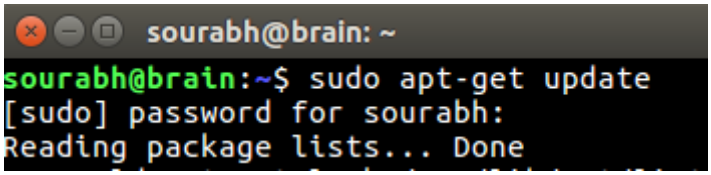
- Prerequisites
- Installation
- Configuring Hadoop in single mode
- Start Hadoop processes
- Viewing map reduce UI
- Multi-node Hadoop cluster setup
- Notes on Hadoop processes

2.1 - PREREQUISITES:

This chapter describes how to set up and configure a single node Hadoop installation so that it can be used to perform simple operations using Hadoop MapReduce and the Hadooop Distributed File System(HDFS).

The prerequisites for installation are: [1]

- SUPPORTED PLATFORMS:
  - GNU/LINUX is supported as a production and development platform.
  - Windows is also supported but the setup procedure here is only for linux.
- REQUIRED SOFTWARE:
  - JAVA must be installed java7 is recommended.
  - Ssh must be installed and sshd must be running to use the Hadoop scripts that manage remote Hadoop daemons.
- A recent stable release of Hadoop distribution for installation.
- NOTE:- [2]
- Hadoop follows master-slave model.
- There can be only 1 master and several slaves.
- In pseudo mode, single node is acting both as master and slave.
- Master machine is also referred to as NameNode Machine.
- Slave machines are also referred to as DataNode machine.
- 2.2 – **INSTALLATION:**
- The method of installation of Hadoop can be implemented to install any version of Hadoop 2.x.x. Since Hadoop requires JVM to run so we need to install Java before installing Hadoop.
- 2.2.1 – **JAVA installation:**
- Package list is update using command:
- $ sudo apt-get update



- **Figure 2.1 – Checking for package update**

- Now java is to be installed on the system, this is accomplished by the command:
- $ sudo apt-get install default-jdk. (Any other command is also acceptable to install JVM).
- After this Java will be installed on the system, to check Java version type :
- $ java –version

```
sourabh@brain:~$ java -version
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
```

- **Figure 2.2 – verifying java-version**

- 2.2.2 – SSH installation:
- SSH is Secure shell. This application allows us to get remote access of any machine(or local host) by different password other than root also allows us to bypass the password by setting it to empty. To install ssh use the following command.
- $ sudo apt-get install ssh

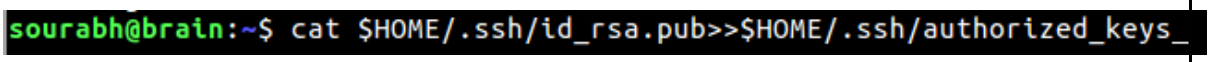
```
sourabh@brain: ~
sourabh@brain:~$ sudo apt-get install ssh_
```

- **Figure 2.3 – Installing SSH**

- If we try to connect local host or local machine through ssh it will ask user for password. To check this a terminal command is :
- $ ssh localhost
- Output :

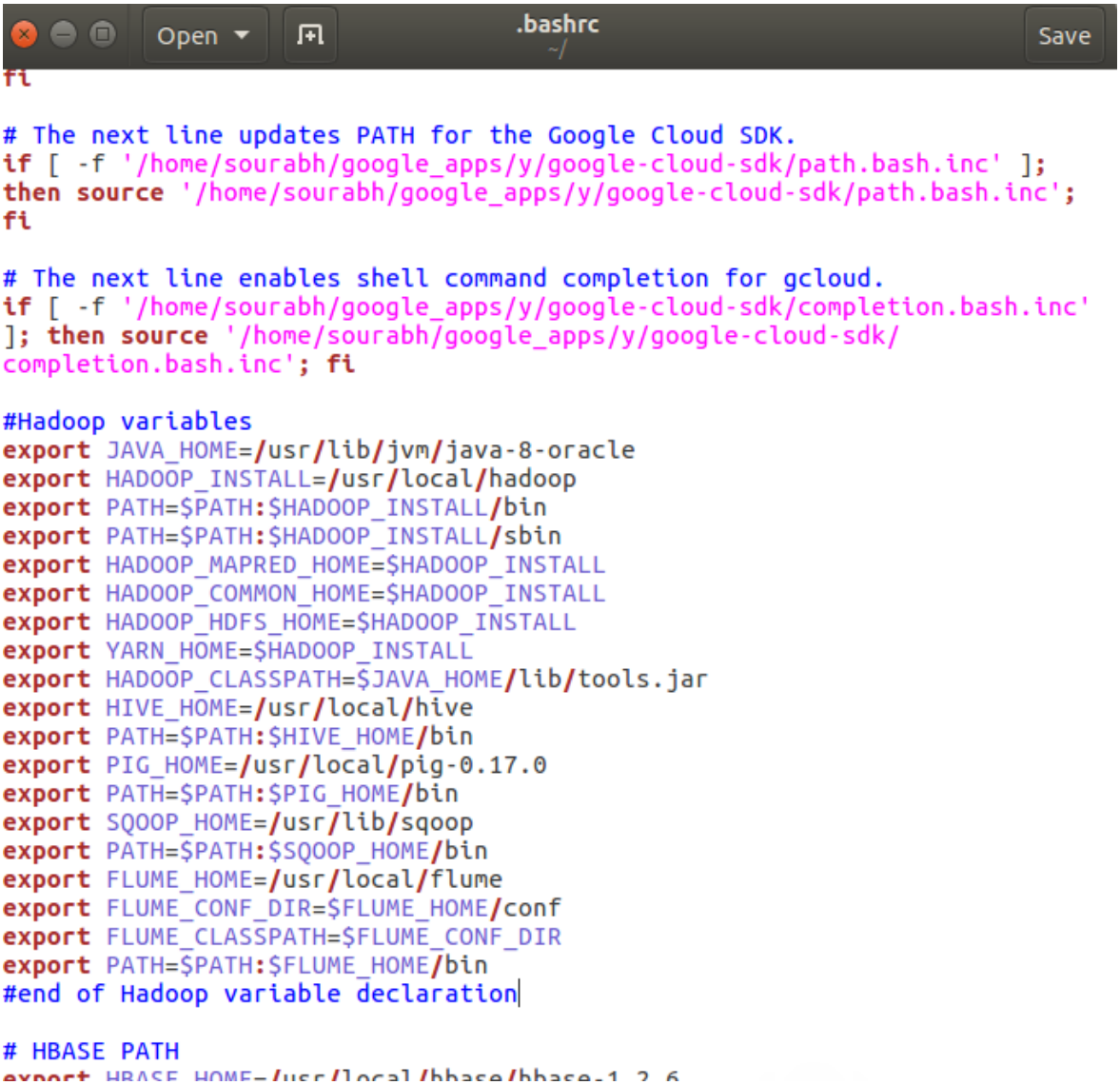
```
sourabh@brain: ~
sourabh@brain:~$ ssh localhost
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-96-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

13 packages can be updated.
13 updates are security updates.

Last login: Tue Jul 18 11:11:40 2017 from 127.0.0.1
```

- **Figure 2.4 – connecting to local host**

- Note:To exit ssh command is ‘exit’
- To set SSH for password less communication the following command is executed on the terminal.
- $ ssh-keygen –t rsa –p’’
- This command will then ask – “Enter file in which to save the key(/home/Sourabh/.ssh/id_rsa):”
- Press enter without typing any single word.
- A image will be then generated, it is called randomart image.
- This key is used to communicate between any two machine for authentication.
- This command creates an RSA key pair with an empty password.(Just because we do not want to enter the passphrase every time Hadoop interacts with its nodes, however it is not recommended to use empty password)
- Now the generated key is to be saved on the local machine’s host key fingerprint to the user’s known host file. The terminal command for this is:
- $cat $HOME/.ssh/id_rsa.pub>>$HOME/.ssh/authorized_keys
- 

    - **Figure 2.5 – saving ssh key on the local file system**
- Now to check that we have bypassed the password, once again we execute the command :
- $ ssh localhost
- Now the system has the all the components required to run Hadoop successfully.
- Once we have the stable Hadoop distribution we need to follow the following steps:
- Extract the ‘hadoop tar.gz’ manually or through terminal.
- Move Hadoop folder to root this step is optional but its recommend that you may move file to root.
- To move Hadoop folder to its appropriate location use following command (note this command is only use to move folder to root if you are placing to other location you can do it manually).
- $sudo mv Desktop/hadoop-2.7.2 /usr/local/Hadoop
- Explanation of the command:
    - -sudo : This is keyword which allows user to grant super user permission temporary. This command is Linux native command. It means “super user do”.

    - -mv : This is Linux native command to move any file or directory to any location. it has two parameters as
    - parameter 1: source Address
    - parameter 2:destination Address Note both above address should be complete qualified address.
- After this, system environment variable has to be set so that our system identifies Hadoop.
- To do this bashrc file is to be edited.
- Procedure for editing bashrc file:
- $sudo gedit ~/.bashrc
- The following lines where appended to the end of bashrc.
- #Hadoop variables
- export JAVA_HOME=/usr/lib/jvm/java-8-oracle
- export HADOOP_INSTALL=/usr/local/hadoop
- export PATH=$PATH:$HADOOP_INSTALL/bin
- export PATH=$PATH:$HADOOP_INSTALL/sbin
- export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
- export HADOOP_COMMON_HOME=$HADOOP_INSTALL
- export HADOOP_HDFS_HOME=$HADOOP_INSTALL
- export YARN_HOME=$HADOOP_INSTALL
- export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
- export HIVE_HOME=/usr/local/hive
- export PATH=$PATH:$HIVE_HOME/bin
- export PIG_HOME=/usr/local/pig-0.17.0
- export PATH=$PATH:$PIG_HOME/bin
- export SQOOP_HOME=/usr/lib/sqoop
- export PATH=$PATH:$SQOOP_HOME/bin
- export FLUME_HOME=/usr/local/flume
- export FLUME_CONF_DIR=$FLUME_HOME/conf
- export FLUME_CLASSPATH=$FLUME_CONF_DIR
- export PATH=$PATH:$FLUME_HOME/bin
- #end of Hadoop variable declaration



- **Figure 2.6 - bashrc**

- Explanation of the lines appended above.
- Line 1: export JAVA_HOME=/usr/lib/jvm/java-8-oracle We are setting Java installation path so that Hadoop can use this path where ever required.
- Line 2: export HADOOP_INSTALL=/usr/local/hadoop this line is to identify installed location of Java in the system. Note if you have kept this folder in some other location you need to change path accordingly.
- Save and close this ~/.bashrc.
- To execute the bashrc file following command is executed on the terminal.
- $ source ~/.bashrc
- After this the system is configured and Hadoop is installed on it.

- To check if Hadoop is successfully installed or not following command is executed on the terminal.
- $ Hadoop version
-

```
sourabh@brain: ~
sourabh@brain:~$ Hadoop version
Hadoop: command not found
sourabh@brain:~$ hadoop version
Hadoop 2.7.2
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r b165c4fe8a74265
c792ce23f546c64604acf0e41
Compiled by jenkins on 2016-01-26T00:08Z
Compiled with protoc 2.5.0
From source with checksum d0fda26633fa762bff87ec759ebe689c
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-2
.7.2.jar
```

- **Figure 2.7 – hadoop version check**
- The next thing is to be done is updating Hadoop environment file.
- We need to update JAVA_HOME in Hadoop .
- Navigate where Hadoop is installed and update the env.sh file located in its 'etc' folder.
- export JAVA_HOME=/usr/lib/jvm/java-8-oracle
-

```
hadoop-env.sh (/usr/local/hadoop/etc/hadoop) - gedit
Open                                                    Save
# See the License for the specific language governing permissions and
# limitations under the License.

# Set Hadoop-specific environment variables here.

# The only required environment variable is JAVA_HOME.  All others are
# optional.  When running a distributed configuration it is best to
# set JAVA_HOME in this file, so that it is correctly defined on
# remote nodes.

# The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java-8-oracle

# The jsvc implementation to use. Jsvc is required to run secure datanodes
# that bind to privileged ports to provide authentication of data transfer
# protocol.  Jsvc is not required if SASL is configured for authentication
of
# data transfer protocol using non-privileged ports.
#export JSVC_HOME=${JSVC_HOME}

export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"}

# Extra Java CLASSPATH elements.  Automatically insert capacity-scheduler.
for f in $HADOOP_HOME/contrib/capacity-scheduler/*.jar; do
  if [ "$HADOOP_CLASSPATH" ]; then
    export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$f
  else
    export HADOOP_CLASSPATH=$f
  fi
done

# The maximum amount of heap to use, in MB. Default is 1000
```

- **Figure 2.8 – editing env.sh file of hadoop**

-
-
- As in my case…/hadoop/etc/hadoop/hadoop-env.sh
- Save and exit.
- Now Hadoop is installed on the system.
- Now Hadoop can be use in three different ways
- 1) Stand Alone Mode: This mode generally does not requires any configuration to be done. This mode is usually used for Debugging purpose. All default configuration of Hadoop are done in this mode.
- 2) Pseudo Distributed Mode: This mode is also called single node mode. This mode needs little configuration. This mode is used for Development purpose
- 3)Distributed Mode: This mode is also called as Multinode node. This mode needs some changes to be done in Psedudistrbuted mode along with ssh This mode is generally use for commercial purpose.
- 2.3 – **CONFIGURING HADOOP IN SINGLE MODE/PSEUDO DISTRIBUTED MODE IN LINUX**:
- Hadoop is by default is configured in Standalone mode. This stand alone mode is used only for debugging purpose but to develops any application we need to configure Hadoop in Pseudo Distributed mode.
- To configure hadoop in Pseudo Distributed mode we need to edit following files:
- 1)core-site.xml
- 2)hdfs-site.xml
- 3)mapred-site.xml
- 4)yarn-site.xml
- All mentioned files are present in hadoop installation directory under "/etc/hadoop" in my case as per previous document its address is "/usr/local/hadoop/etc/hadoop".
- 2.3.1 – **CONFIGURING CORE-SITE.XML**
- core site xml is a file containing all core property of Hadoop. For example. Namenode url, Temporary storage directory path, etc. Hadoop has predefined configuration which we need to override them if we mention any of the

configuration in core-site.xml then during startup of Hadoop, Hadoop will read these configuration an run Hadoop using this.

- This file can be edited in any text editor.
- File is opened in any of the text editor and these contents where added in it between.
- <configuration>
- <property>
- <name>fs.defaultFS</name>
- <value>hdfs://localhost:9000</value>
- </property>
- <property>
- <name>hadoop.tmp.dir</name>
- <value>/home/sourabh/tmp</value>
- </property>
- </configuration>

core-site.xml (/usr/local/hadoop/etc/hadoop) - gedit

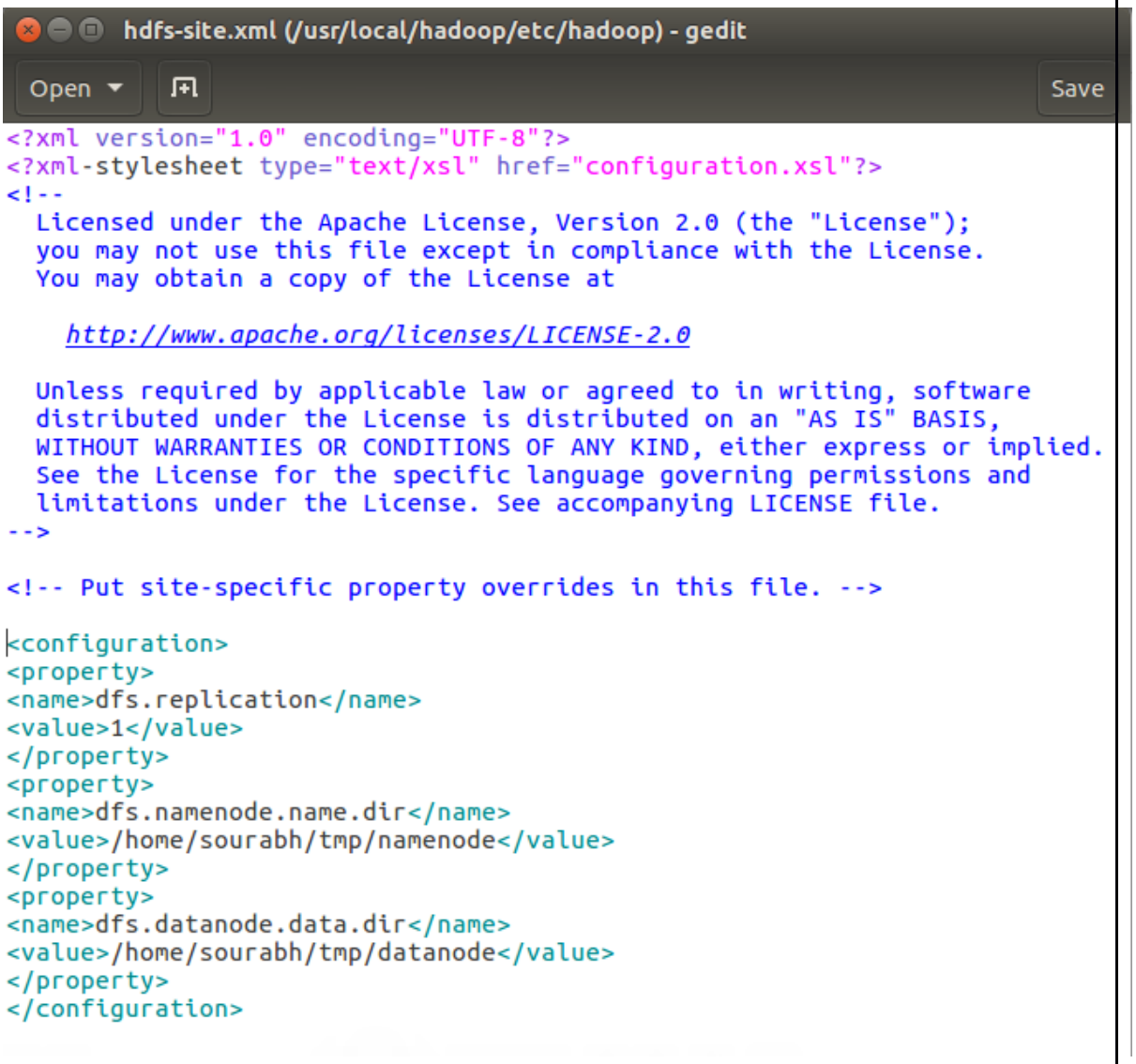
Open Save

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
<name>fs.defaultFS</name>
<value>hdfs://localhost:9000</value>
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>/home/sourabh/tmp</value>
</property>
</configuration>
```

- **Figure 2.9 – core-site.xml**

- Explanation of the above code:
- property 1: fs.defaultFS
    - This property overrides the default namenode url its syntax is hdfs://: .This property was named as fs.default.name in hadoop 1.x.x version. Note: Port number can be any number above 255 to 65536
- property 2: hadoop.tmp.dir
    - This property is used to change the temporary storage directory during execution of any algorithm in hadoop by default its location is "/tmp/hadoop-${user.name}" in my case I have created this directory in my home folder name tmp so its "/home/sourabh/tmp".
- 2.3.2 – **CONFIGURING HDFS-SITE.XML**

- This file contains all configuration about hadoop distributed file system also called as HDFS such as storage location for namenode, storage location for datanode, replication factor of HDFS, etc. Similar to core-site.xml we need to place below content between configuration fields.
- <configuration>
- <property>
- <name>dfs.replication</name>
- <value>1</value>
- </property>
- <property>
- <name>dfs.namenode.name.dir</name>
- <value>/home/sourabh/tmp/namenode</value>
- </property>
- <property>
- <name>dfs.datanode.data.dir</name>
- <value>/home/sourabh/tmp/datanode</value>
- </property>
- </configuration>



- **Figure 2.10 hdfs-site.xml**

- Explanation of above properties in detail.
- Property 1: dfs.replication: This property overrides the replication factor in hadoop. By default its value is 3 but in single node cluster it is recommended to be 1.
- Property 2: dfs.namenode.name.dir: This property overrides storage location of namenode data by default its storage location is inside "/tmp/hadoop-${user.name}". To change this you have set value of your folder location in my case it is inside tmp directory created during core-site.xml
- Property 3: dfs.datanode.data.dir: This property overrides storage location of datanode data by default its storage location is inside "/tmp/hadoop-${user.name}". To change this you have set value of your folder location in my case it is also inside tmp directory created during core-site.xml

- 2.3.3 - **CONFIGURING MAPRED-SITE.XML**
- This file contain all configuration about Map Reduce component in hadoop. Please note that this file doesn't exist but you can copy or rename it from mapred-site.xml.template. Configuration for this file is should be as followed.
- <configuration>
- <property>
- <name>mapreduce.framework.name</name>
- <value>yarn</value>
- </property>
- </configuration>
-

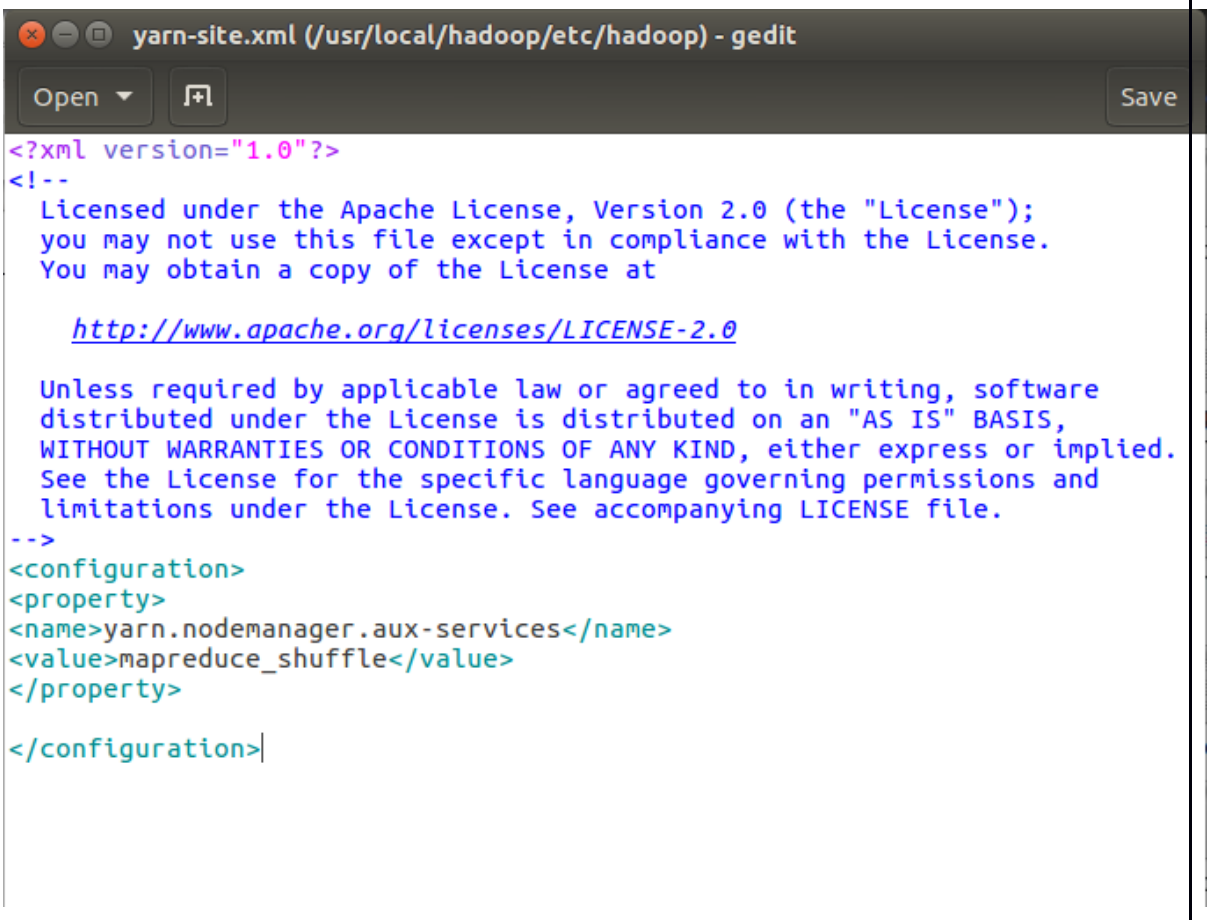
-

- **Figure 2.11 – mapred-site.xml**

- Explanation of above property
- Hadoop 2.x.x hadoop has introduced new layer of technology developed by hadoop to improve performance of map reduce algorithm this layer is called as "yarn" that is Yet Another Resource Negotiator. So here we are configuring that

our hadoop framework is yarn if we don't specify this property then our hadoop will use Map reduce 1 also called as MR1.

- 2.3.4 - **CONFIGURING YARN-SITE.XML**
- This file contains all information about YARN as we will be using MR2 we need to specify the auxiliary services that need to be used with MR2 so add these lines to yarn-site.xml
- <property>
- <name>yarn.nodemanager.aux-services</name>
- <value>mapreduce_shuffle</value>
- </property>



- **Figure 2.12 – yarn-site.xml**

2.3.5 - CREATING HDFS

Now Hadoop has been configured in Pseudo distributed mode. Before starting Hadoop namenode is to be formatted.

Terminal command for this is:

$hdfs namenode –format

This command will create hadoop-temp directory.

2.4 - START HADOOP PROCESSES

Now to start Hadoop:

$ start-all.sh

To check which components are working terminal command is:

$ jps

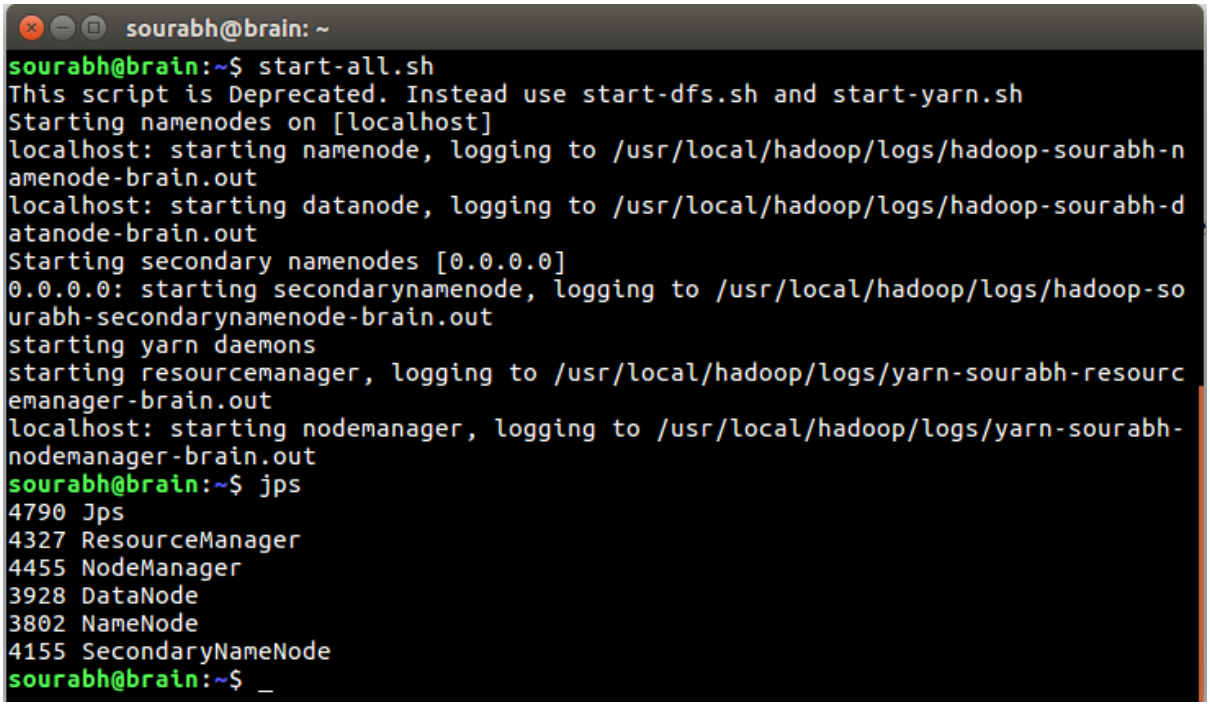


Figure 2.13 – Hadoop cluster running

2.5 - VIEWING MAP REDUCE UI

The map-reduce UI can be viewed by typing localhost:50070 in the browser.

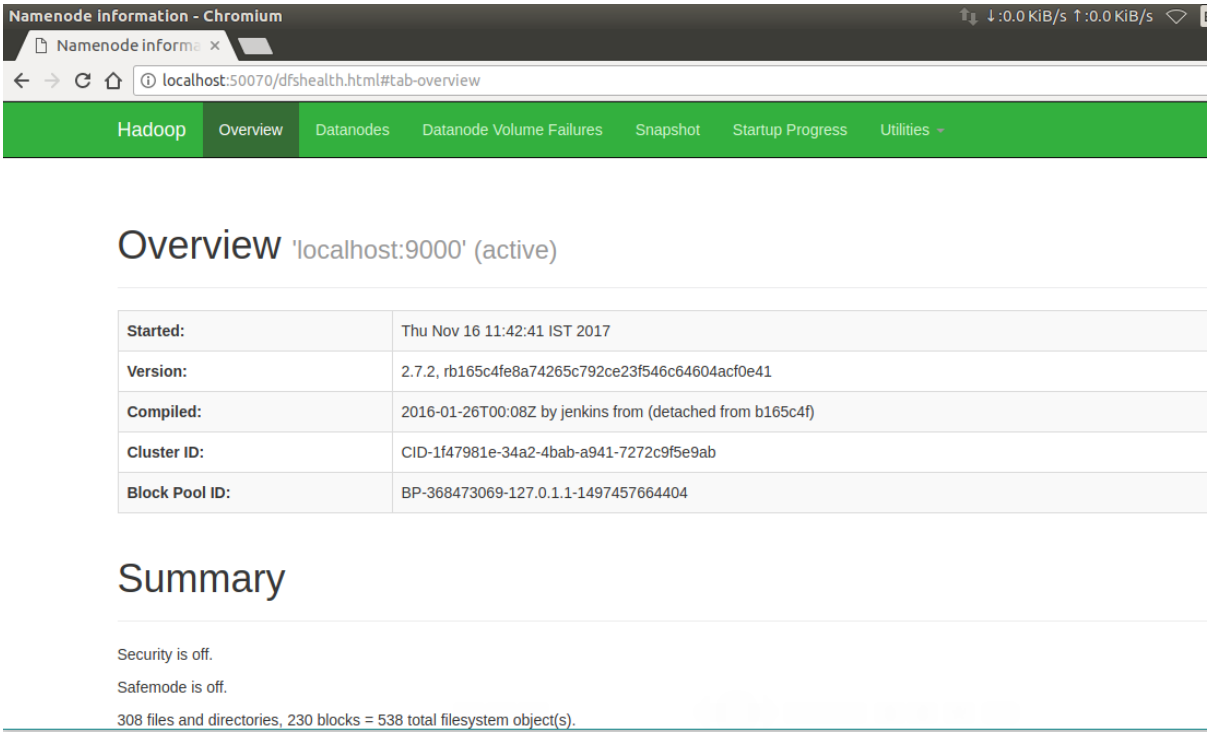


Figure 2.14 – local host of hadoop

2.6 - MULTI-NODE HADOOP CLUSTER SETUP

The standalone mode is used only for debugging purposes but for developing any application we need to configure Hadoop in distributed mode.

To configure hadoop in Pseudo Distributed mode we need to edit following files

1) Add all Slaves/ Other Machines (Datanode, Job tracker, node Manager) to /etc/hosts.

2) Copy SSH key from Master (Namenode) to Slave file.

3) Edit core-site.xml

4) Edit hdfs-site.xml

5) Edit yarn-site.xml

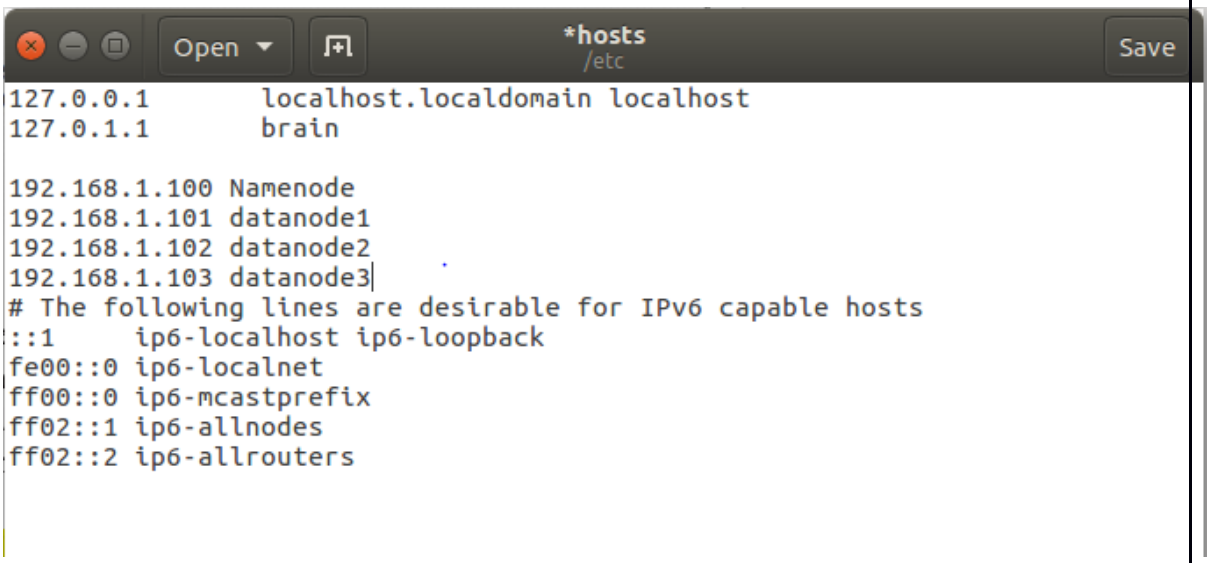
6) Format Cluster

7) Start Cluster

2.6.1 - ADD ALL SLAVES TO HOSTS FILE

- Authorized_keys file has to be copied into all the machines.
- Make sure you can do ssh to all the machines in a password less manner.

- Adding Other Machines (Slaves to /etc/hosts) to do that open host file on Namenode by below command and list down all slave machine to it
- $sudo gedit /etc/hosts



- **Figure 2.15 – host file**

- **2.6.2 - <u>COPY SSH KEYS TO ALL SLAVES FILES(NAMENODE ONLY)</u>**
- This will allow our namenode to communicate our Datanode using password less communication. Command for it is
- $ ssh-copy-id -i $HOME/.ssh/id_rsa.pub user-name@pc-name

NOTE: The user name is slave user account on which you want to run hadoop and pc-name is name of slave you have entered in step 1 in /etc/hosts file.

Do same process for each Datanode.

2.6.3 - <u>EDIT CORE-SITE.XML</u>

Core site xml is a file containing all core property of hadoop. For example. Namenode url, Temporary storage directory path, etc. Hadoop has predefined configuration which we need to override them if we mention any of the configuration in core-site.xml then during start-up of hadoop, hadoop will read these configuration a run hadoop using this.

<configurations></configurations>

<property>

<name>fs.default.name</name>

<value>hdfs://Namenode:9000</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/home/sourabh/tmp</value>

</property>

<property>

<name>dfs.permissions</name>

<value>false</value>

</property>

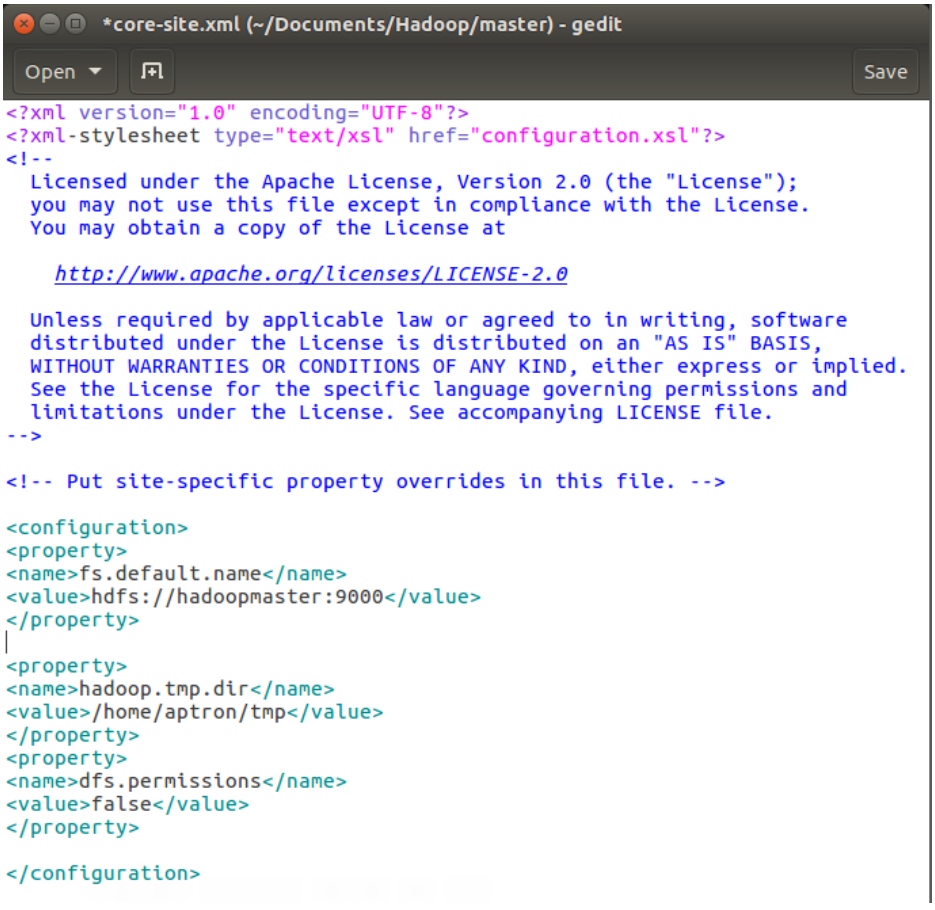


Figure 2.16 – cluster setup file

2.6.4 - EDIT HDFS-SITE.XML

This file contains all configuration about hadoop distributed file system also called as HDFS such as storage location for Namenode, storage location for Datanode,

replication factor of HDFS, etc. Similar to core-site.xml we need to place below content between configuration.

(FOR BOTH NAMENODE AND DATANODE)

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

2.6.5 - EDIT YARN-SITE.XML

This file contains all information about YARN as we will be using MR2 we need to specify the auxiliary services that need to be used with MR2 so add these lines to yarn-site.xml.

(For Namenode/Master Node)

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce_shuffle</value>

</property>

(For Slave/ Datanode)

<property>

<name>

yarn.nodemanager.aux-services.mapreduce.shuffle.class

</name>

<value>org.apache.hadoop.mapred.shuffleHandler</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>192.168.1.100:9025</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address</name>

<value>192.168.1.100:9030</value>

</property>

<property>

<name>yarn.resourcemanager.address</name>

<value>192.168.1.100:9040</value>

</property>

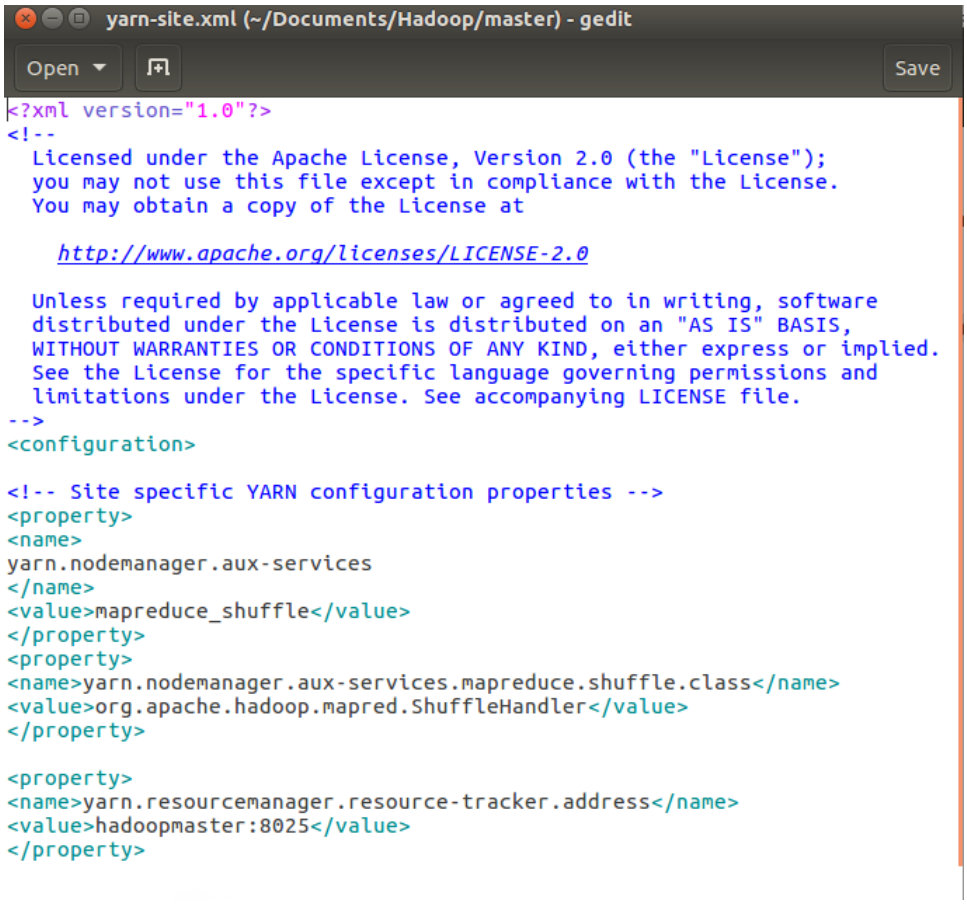


Figure 2.17 – cluster yarn-site.xml

2.6.6 - FORMAT CLUSTER

Before staring Hadoop we need to format our namenode. Execute this command to format namenode.

$ sudo namenode-format

2.6.7 - START CLUSTER

Now to start cluster the terminal command is

$ start all.sh

2.6.8 - CONFIGURING MASTER AND SLAVES FILES

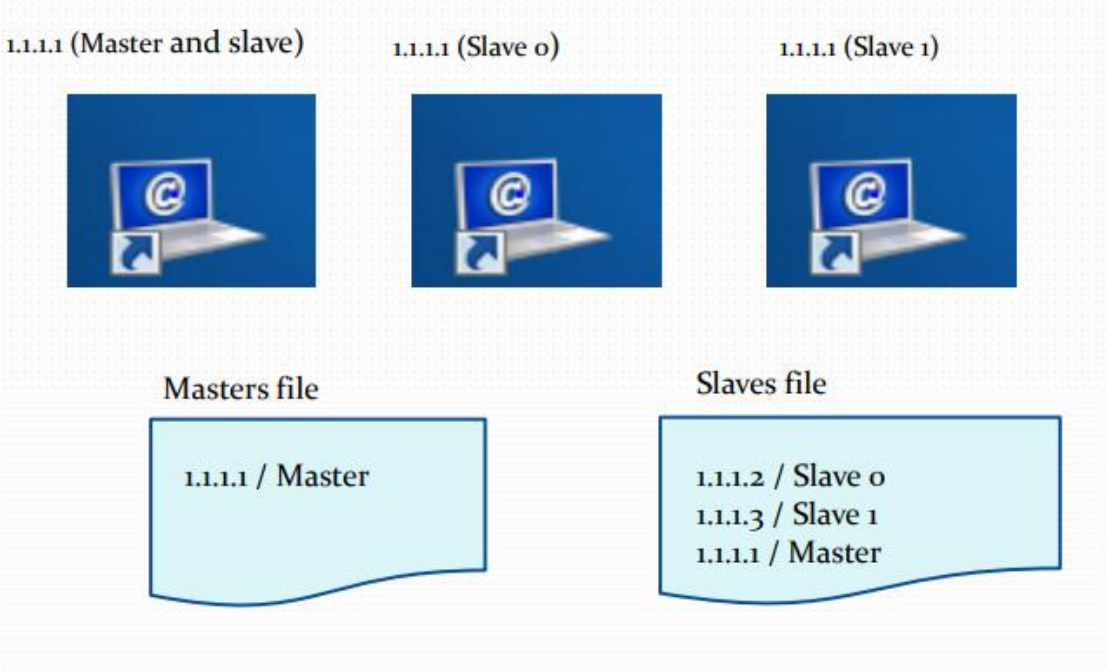


Figure 2.18 - MASTER ACTING AS SLAVE:

CHAPTER 3 Directory Structure of Hadoop

An Overview

- /sbin
- /bin
- /etc
- /lib
- /logs
- /include
- /share

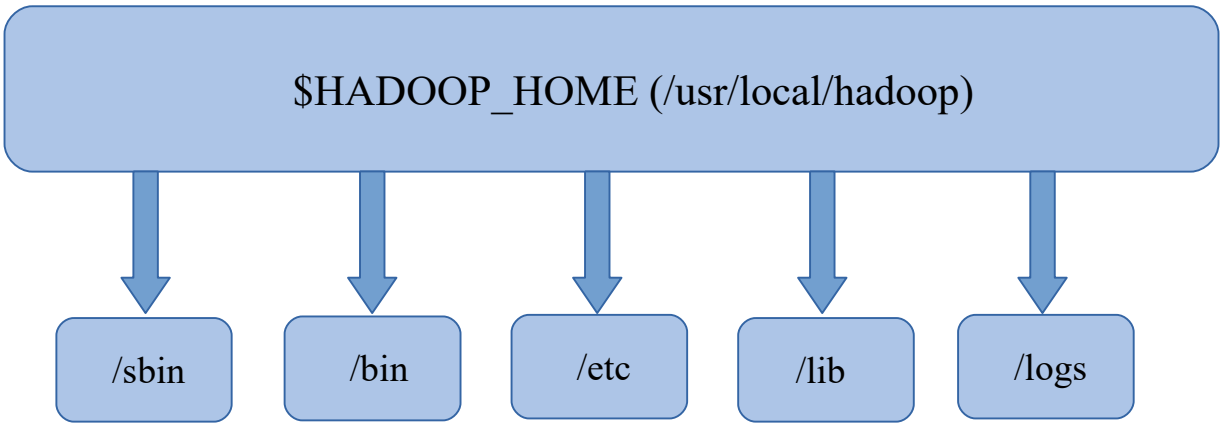


Figure - 3.1 Hadoop Directory Structure

3.1 - /sbin

This folder contains all the files that are required for starting the Hadoop application.

It consists of the scripts like:

- start-dfs.sh

  It starts the HDFS services of Hadoop.

- start-yarn.sh

  It starts the YARN services.

- start-all.sh

  It is depricated in the Hadoop 2.x since its does the same work as the start-dfs.sh and start-yarn.sh combined.

- stop-all.sh

  It is used to stop all the Hadoop services currently running on the node.

3.2 - /bin

In Hadoop 1.x, bin folder used to contain all the Hadoop executable files but after the update to Hadoop 2.x, the services and commands specific to initiating and running the cluster were moved to sbin folder (which makes sense as the sbin folders do contain the boot files for an application).

It consists of the scripts for:

- hadoop

- hdfs
- mapred
- yarn

3.3 - /etc

etc folder further contains “hadoop/” folder.

Within this folder are placed all the configuration files.

All the Hadoop related properties need to go into one of these files:

- mapred-site.xml
- core-site.xml
- hdfs-site.xml
- yarn-site.xml

3.4 - /lib

All the third party jar files that one would require while working with the HDFS API are present in this location.

These could range from compression algorithms to the MapReduce job optimization.

3.5 - /logs

All the log files for the corresponding process will be created here.

These logs include:

- NameNode logs
- DataNode logs
- Secondary NameNode logs
- Job Tracker logs
- Task Tracker logs

These were the five main sub directories in the Hadoop folder. Besides these, the Hadoop directory also contains other directories like:

3.6 - /include

All the third party header files required for including in the main MapReduce jobs.

3.7 - /share

All the documentation and the extra programs that are frequently used in the MapReduce jobs are present in this directory.

It contains the data on the default properties values and the all the other important information one needs to learns to properly work with Hadoop.

Also the programs like Word counting, arranging, sorting and many more are given as jar files.Except these folders, Hadoop directory also contains files like:

- LICENSE.txt
- NOTICE.txt
- README.txt

These files contain the disclaimer and license information about the Apache software.

CHAPTER 4 IMPORTANT CONFIGURATON PROPERTIES

An Overview

- fs.default.name
- mapred.job.tracker.
- dfs.block.size.
- dfs.replication.
- Hadoop.tmp.dir.
- Namespace id.
- Safemode.

4.1 – fs.default.name

Hadoop fs.default.name configuration property is used to specify the file system. hadoop fs.default.name variable is used to specify the default file system. The URI scheme and authority specified in this property determines the file system implementation. URI scheme determines the filesystem implementation class and the authority determines the host name and port number of the file system.

In hadoop, the fs.default.name property is used to specify the namenode machines hostname and port number. This property needs to be set in core-site.xml file.

- The name of the default file system. A URI whose scheme and authority determine the FileSystem implementation. The uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority is used to determine the host, port, etc. for a filesystem.
- Value is “hdfs://IP:PORT” [hdfs://localhost:54310].
- Specifies where is your name node is running.
- Any one from outside world trying to connect to Hadoop cluster should know the address of the name node

4.2 – mapred.job.tracker

The host and port that the MapReduce job tracker runs at. If "local", then jobs are run in-process as a single map and reduce task.

Hadoop mapred.job.tracker property is used to set the jobtrackers hostname and port number.

- Value is “IP:PORT” [localhost:54311]
- Specifies where is your job tracker is running.
- When external client tries to run the map reduce job on Hadoop cluster should know the address of job tracker.

4.3 – dfs.block.size

Hadoop Distributed File System is designed to hold and manage large amounts of data; therefore, typical HDFS block sizes are significantly larger than the block sizes for a traditional file system (for example, the file system on my laptop uses a block size of 4 KB). The block size setting is used by HDFS to divide files into blocks and then distribute those blocks across the cluster. For example, if a cluster is using a

block size of 64 MB, and a 128-MB text file was put in to HDFS, HDFS would split the file into two blocks (128 MB/64 MB) and distribute the two chunks to the data nodes in the cluster.

- Default value is 64MB.
- File will be broken into 64MB chunks.
- It is one of the tuning parameters.
- Directly proportional to number of mapper tasks running on Hadoop cluster.

4.4 – dfs.replication

Hadoop distributed file system(HDFS) stores files as data blocks and distributes these blocks across the entire cluster. As HDFS is designed to be fault-tolerant and to run on commodity hardware, blocks are replicated a number of times to ensure high data availability. The replication factor is a property that can be set in the HDFS configuration file that will allows to adjust the global replication factor for the entire cluster. For each block stored in HDFS, there will be n-1 duplicated blocks distributed across the cluster. For example, if the replocation factor was set to 3 there would be one original block and two replicas.

This replication factor can be changed by editing the HDFS-site.xml present in conf/ folder.

- Defines the number of copies to be made for each block.
- Replication features achieves fault tolerant in the Hadoop cluster.
- Data is not lost even if the machines are going down.
- OR it achieves Data Availablity

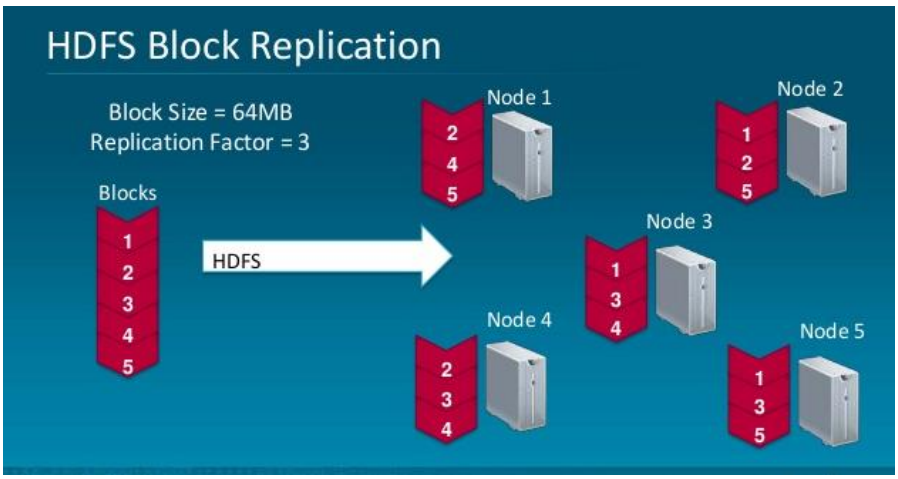


Figure 4.1 – hdfs blocks

4.5 - **hadoop.tmp.dir**

A base for other temporary directories.

- The value of this property is a directory.
- This directory consists of Hadoop file system information.
- Consist of meta data image.
- Consist of blocks etc.

Hadoop.tmp.dir is user as a base for temporary directories locally, and also in HDFS. There are three HDFS properties which contain hadoop.tmp.dir in their values:

- dfs.name.dir: directory where namenode stores its metadata, with default value ${hadoop.tmp.dir}/dfs/name.
- dfs.data.dir: directory where HDFS data blocks are stored, with default value ${hadoop.tmp.dir}/dfs/data.
- fs.checkpoint.dir: directory where secondary namenode store its checkpoints, default value is ${hadoop.tmp.dir}/dfs/namesecondary.

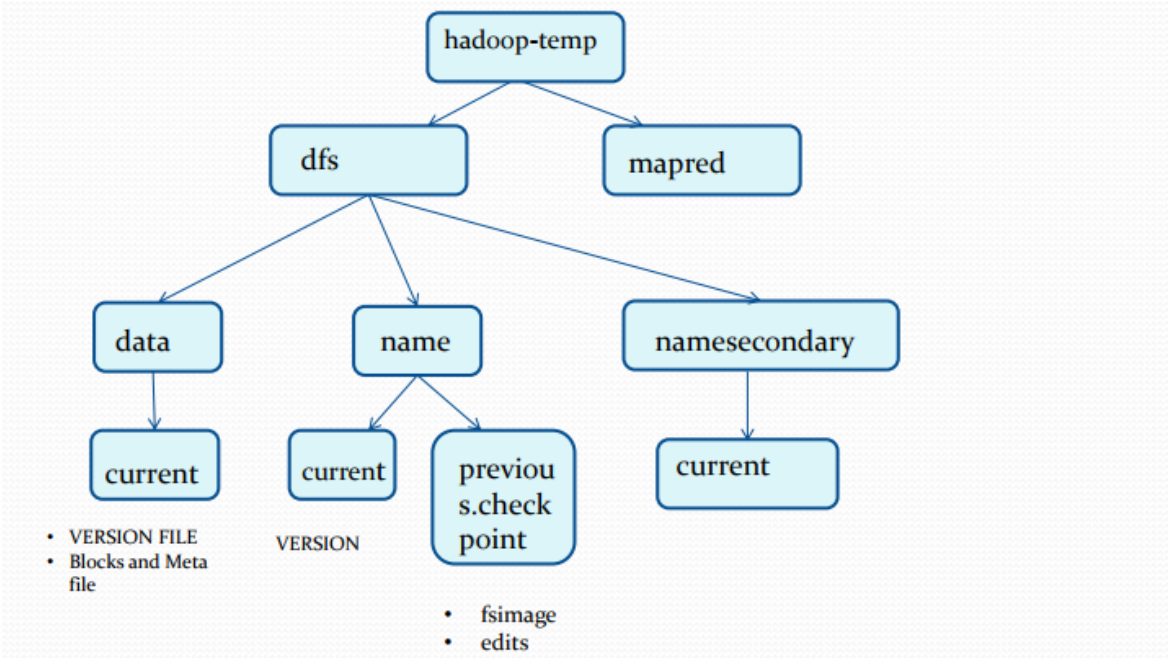
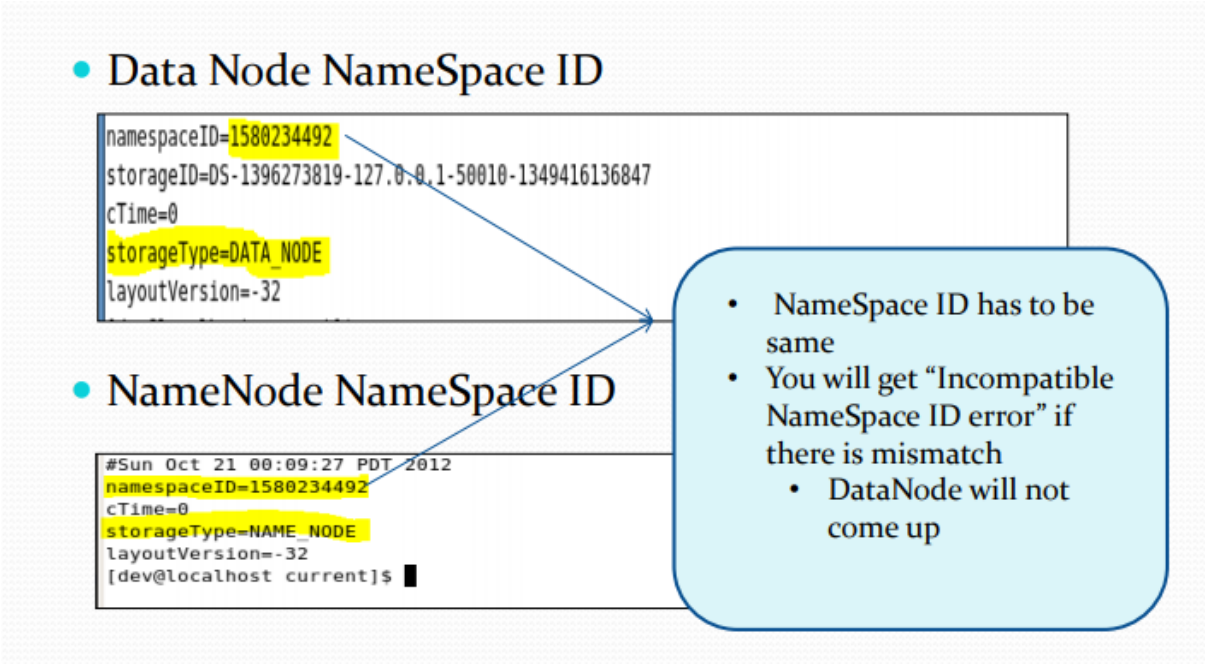


Figure 4.2 - Hadoop directory structure

4.6 - NAMESPACE ID'S



- Every time namenode is formatted, new namespace id is allocated to each of the machines.
- DataNode namespace id has to be matched with NameNode's namespace id.

Formatting the namenode will result into creation of new HDFS and previous data will be lost.

4.7 - SAFEMODE:

A Safemode for Namenode is essentially a read-only mode for the HDFS cluster, where it does not allow any modifications to file system or blocks. Normally, Namenode disables safe mode automatically at the beginning. If required, HDFS could be placed in safe mode explicitly.

- Starting Hadoop is not a single click.
- When Hadoop starts up it has to do lot of activities.
  - Restoring the previous HDFS state.
  - Waits to get the block reports from all the Data Node's etc
- During this period Hadoop will be in safe mode
  - It shows only meta data to the user
  - It is just Read only view of HDFS.
  - Cannot do any file operation.

- Cannot run MapReduce job.

- For doing any operation on Hadoop SafeMode should be off.
- Run the following command to get the status of safemode.
  - $ hadoop dfsadmin -safemode get
- For maintenance, sometimes administrator turns ON the safe mode
  - $ hadoop dfsadmin -safemode enter
- Run the following command to turn off the safemode.

$ hadoop dfsadmin -safemode leave

CHAPTER 5 HDFS SHELL COMMANDS

An Overview

- What is HDFS?
- How to use HDFS Shell commands
- Command to create and list the directories and files
- Command to put files on HDFS from local file system
- Command to put files from HDFS to local file system
- Displaying the contents of file
- Some useful commands Table

5.1 - What is HDFS?

The Hadoop Distributed File System (HDFS) is a sub-project of the Apache Hadoop project. This Apache Software Foundation project is designed to provide a fault-tolerant file system designed to run on commodity hardware.

According to The Apache Software Foundation, the primary objective of HDFS is to store data reliably even in the presence of failures including NameNode failures, DataNode failures and network partitions. The NameNode is a single point of failure for the HDFS cluster and a DataNode stores data in the Hadoop file management system.

HDFS uses a **master/slave** architecture in which one device (the master) controls one or more other devices (the slaves). The HDFS cluster consists of a single NameNode and a master server manages the file system **namespace** and regulates access to files.HDFS is a **filesystem** designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware.[5.1]

HDFS is a layered or Virtual file system on top of local file system but it does not modify the underlying file system.

Each of the slave machine accesses data from HDFS as if they were accessing from their local file system.

HDFS follows **WORM** strategy which stands for "**Write Once Read Many times**". When putting the data on HDFS, it is broken(dfs.block.size), replicated and distributed across several machines (except when in standalone mode or pseudo-distributed mode).

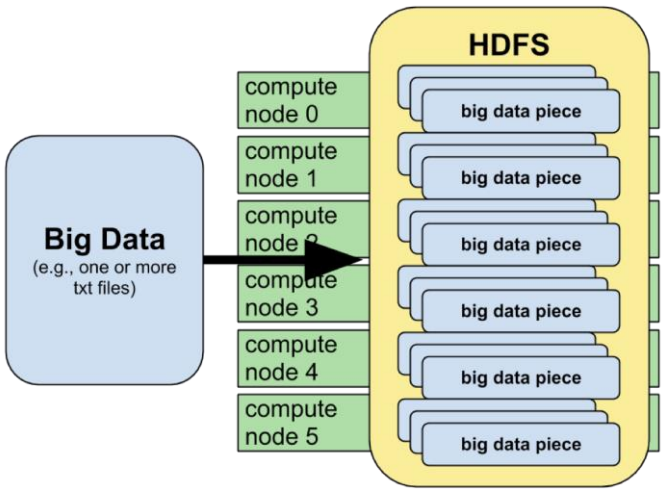


Figure 5.1 HDFS working

Behind the scenes when a file is put in the HDFS,

- File is broken into blocks.
- Each block is replicated.
- Replica's are stored on local file system of data nodes.

Having a block abstraction for a distributed filesystem brings several benefits. The first benefit is the most obvious: a file can be larger than any single disk in the network. There's nothing that requires the blocks from a file to be stored on the same disk, so they can take advantage of any of the disks in the cluster. In fact, it would be possible, if unusual, to store a single file on an HDFS cluster whose blocks filled all the disks in the cluster.

Second, making the unit of abstraction a block rather than a file simplifies the storage subsystem. Simplicity is something to strive for in all systems, but it is especially important for a distributed system in which the failure modes are so varied. The storage subsystem deals with blocks, simplifying storage management (because blocks are a fixed size, it is easy to calculate how many can be stored on a given disk) and eliminating metadata concerns (because blocks are just chunks of data to be stored, file metadata such as permissions information does not need to be stored with the blocks, so another system can handle metadata separately).

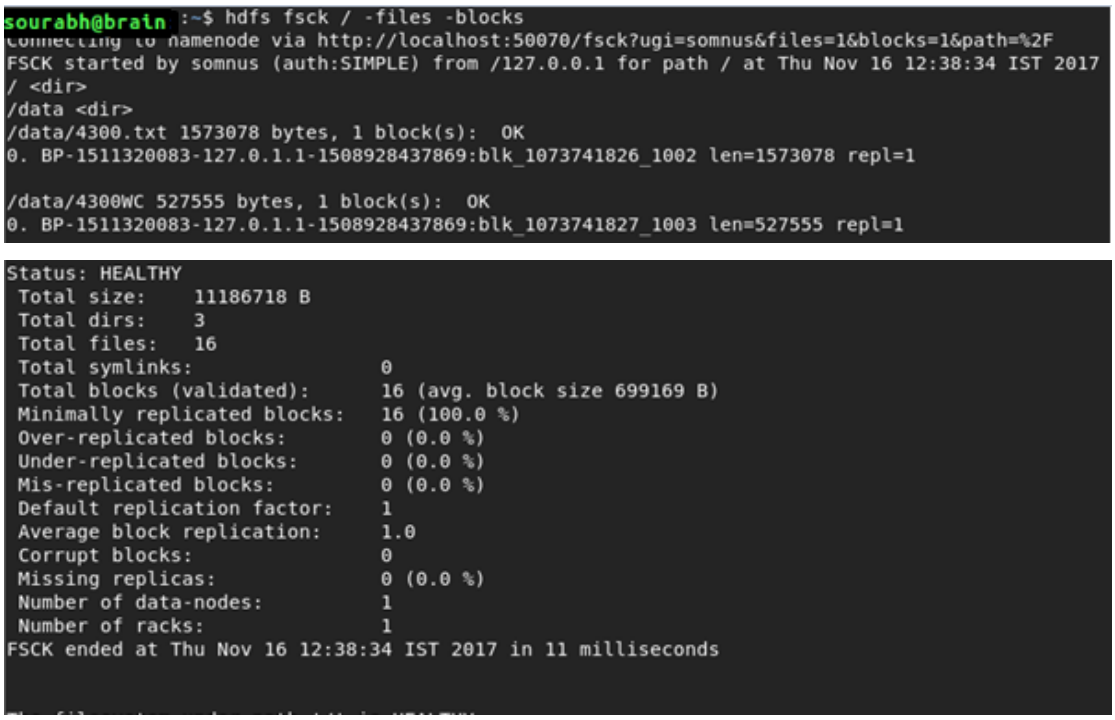


Figure 5.2 HDFS fsck

HDFS understands blocks, therefore when running

% hdfs fsck / -files -blocks

will list the blocks that make up each file in the filesystem.

The Command-Line Interface

We're going to have a look at HDFS by interacting with it from the command line. There are many other interfaces to HDFS, but the command line is one of the simplest and, to many developers, the most familiar.

5.2 - How to use HDFS Shell commands

HDFS is not a regular file system. Therefore, unix commands like cd, ls, mv, etc will not work directly.

For accessing HDFS through command line, we use:

"hadoop fs –options"

"options" are the various commands.

These may or may not be from linux origin. Hence it does not support all of linux commands.

Change directory command "cd" is not supported.

Cannot open a file in HDFS using VI editor or any other editor for editing. Meaning you cannot edit any file residing on HDFS. But as of Hadoop 2.6.x, the data can be appended (cannot seek to any other location in the file) to a file and the appended data has to end before EOF.

The filesystem is ready to be used, and we can do all of the usual filesystem operations, such as reading files, creating directories, moving files, deleting data, and listing directories. we can type

% hadoop fs -help

to get detailed help on every command.

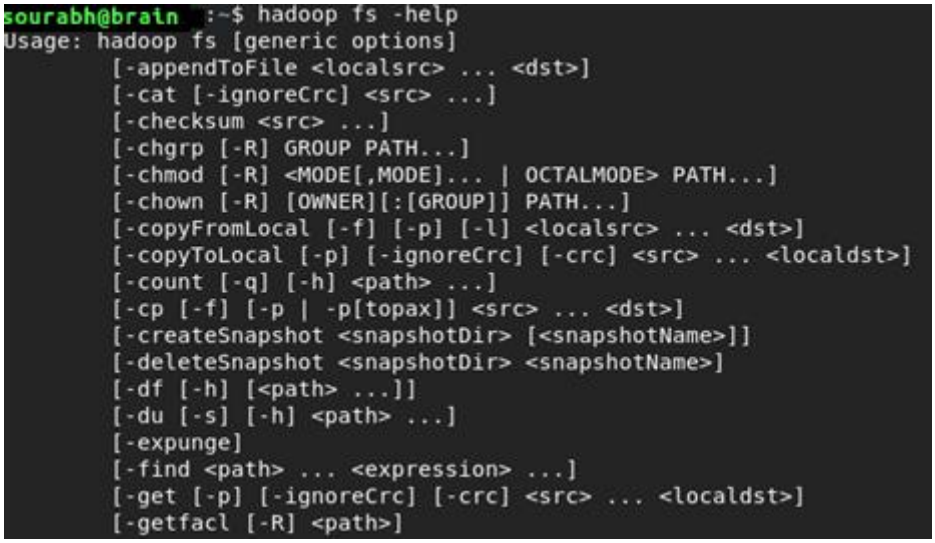


Figure 5.3 few of the available Hadoop command options.

HDFS Home Directory

All the operations like creating files or directories are done in the home directory. All the files and folders of a user are placed inside home directory of linux (/home/user/).

By default, HDFS home directory is

fs.default.name/user/sourabh

hdfs://localhost:54310/user/sourabh/

"sourabh" is the user who has created the filesystem.

"54310" is the port number.

Browsing HDFS

To view and browse HDFS files, we can go to

http://localhost:50070/explorer.html#

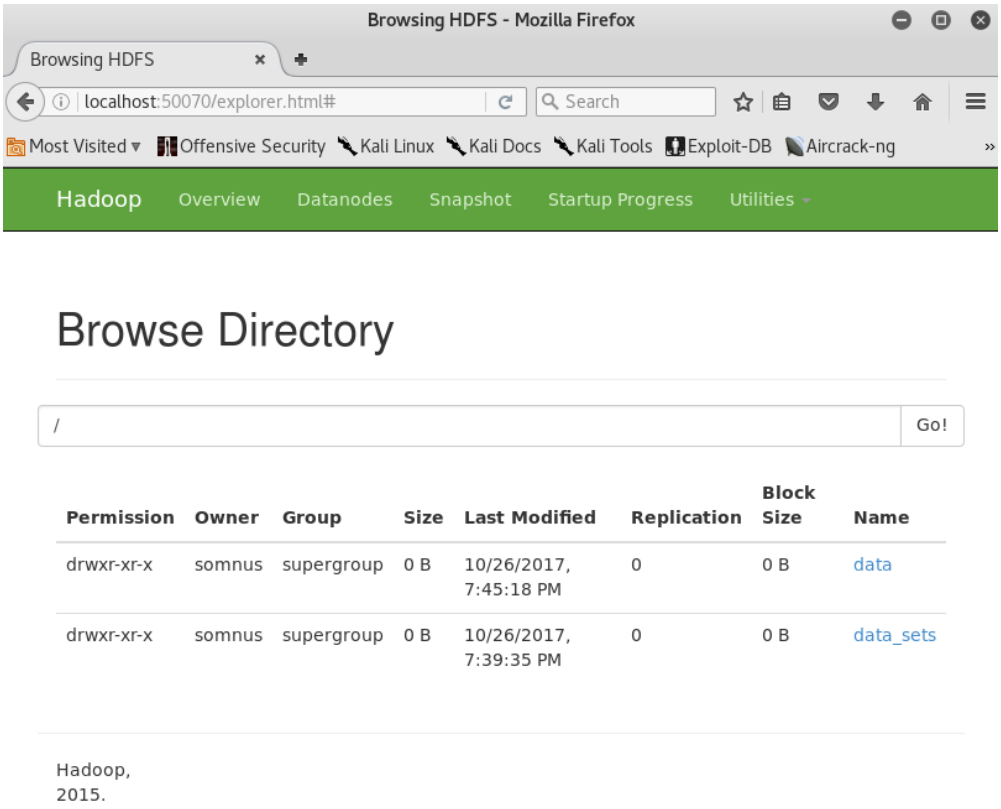


Figure 5.4 Browsing HDFS

All the files and folders that are put in the HDFS will be listed here.

The port number can be changed in the configuration files.

5.3 - Command to create and list the directories and files

5.3.1 % hadoop fs -mkdir /foo

It will create the "foo" directory under HDFS home directory.

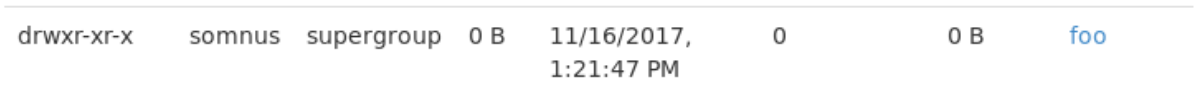


Figure 5.5 folder created in the HDFS

5.3.2 % hadoop fs -ls /

This command is used for same purpose as in linux shell. Listing!

It displays all the necessary details about directories and files.

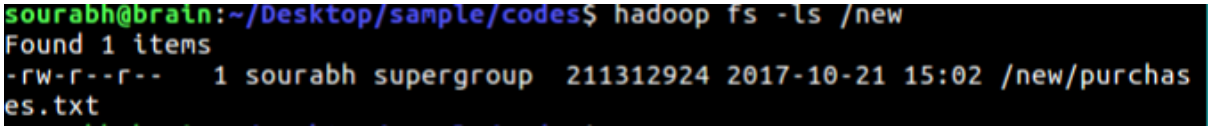


Figure 5.6 HDFS Listing

There are eight columns in the displayed output:

File Mode	Replication factor	File owner	File group	Size	Last Modified date	Last modified time	Absolute name of the file or directory

Figure 5.7 Column in listing

The information returned is very similar to that returned by the Unix command **ls -l** , with a few minor differences. The first column shows the file mode. The second column is the replication factor of the file (something a traditional Unix filesystem does not have). We set the default replication factor in the site-wide configuration to be 1, which is why we see the same value here. The entry in this column is empty for directories because the concept of replication does not apply to them—directories are treated as metadata and stored by the NameNode, not the DataNodes. The third and fourth columns show the file owner and group. The fifth column is the size of the file in bytes, or zero for directories. The sixth and seventh columns are the last modified date and time. Finally, the eighth column is the name of the file or directory.

HDFS has a permissions model for files and directories that is much like the POSIX model. There are three types of permission: the read permission (r), the write permission (w), and the execute permission (x). The read permission is required to read files or list the contents of a directory. The write permission is required to write a file or, for a directory, to create or delete files or directories in it. The execute permission is ignored for a file because you can't execute a file on HDFS (unlike POSIX), and for a directory this permission is required to access its children.

5.4 - Command to put files on HDFS from local file system

5.4.1 % hadoop fs -copyToLocal /src_hdfs /dest_local

This would copy files to the local file system, retaining the original in the HDFS.

5.4.2 % hadoop fs -get /src_hdfs /dest_local

This is another variant of -copyToLocal command.

5.5 - Command to put files from HDFS to local file system

5.5.1 % hadoop fs -copyFromLocal /src_local /dest_hdfs

Copies the file from local file system to HDFS home directory by name foo

5.5.2 % hadoop fs -put /src_local /dest_hdfs

Another variation to -copyFromLocal, -put is a bit different from -get as in this case, the original in the local file system is not maintained but moved to the HDFS.

5.6 - Displaying the contents of file

% hadoop fs -cat /hdfs_loc

It is used to display the contents of the files on the HDFS, just like the "cat" command in the linux, directly in the terminal.

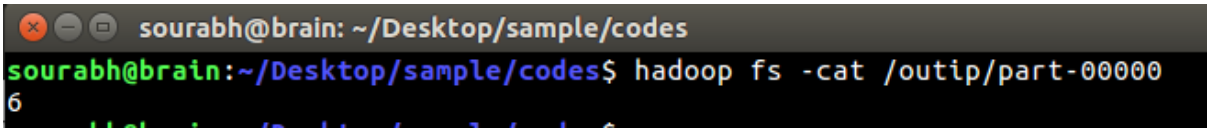


Figure 5.8 Displaying contents of outip in terminal.

Besides these commands there are many others that can be used.

5.7 - Some useful HDFS commands (Table 5.1)

1. **lsr <path>**

   Behaves like -ls, but recursively displays entries in all subdirectories of path.

2. **du <path>**

   Shows disk usage, in bytes, for all the files which match path; filenames are reported with the full HDFS protocol prefix.

3. **dus <path>**

   Like -du, but prints a summary of disk usage of all files/directories in the path.

4. **mv <src><dest>**

   Moves the file or directory indicated by src to dest, within HDFS.

5. **cp <src> <dest>**

   Copies the file or directory identified by src to dest, within HDFS.

6. **rm <path>**

Removes the file or empty directory identified by path.

rmr <path>

7. Removes the file or directory identified by path. Recursively deletes any child entries (i.e., files or subdirectories of path).

getmerge <src> <localDest>

8. Retrieves all files that match the path src in HDFS, and copies them to a single, merged file in the local file system identified by localDest.

setrep [-R] [-w] rep <path>

9. Sets the target replication factor for files identified by path to rep. (The actual replication factor will move toward the target over time)

touchz <path>

10. Creates a file at path containing the current time as a timestamp. Fails if a file already exists at path, unless the file is already size 0.

CHAPTER 6 HDFS COMPONENTS

An Overview

- Introduction to HDFS.
- Benefits of HDFS.
- Working of NameNode.
- Working of DataNode.
- Working of secondary NameNode.
- YARN.
- Writing a file on HDFS.
- Reading a file from HDFS.

6.1 - INTRODUCTION TO HDFS

Hadoop Distributed File System or HDFS is a distributed storage space which spans across an array of commodity hardware. This file system is stable enough to handle any kind of fault and has an efficient throughput which the stream data can access in an efficient and reliable manner. The architecture of HDFS is suitable to store large volume of data. HDFS can process data very rapidly. HDFS is a part of Apache Hadoop eco-system.

The Apache Hadoop framework is composed of the following modules:

- Hadoop Common – The common module contains libraries and utilities which are required by other modules of Hadoop.
- Hadoop Distributed File System (HDFS) – This is the distributed file-system which stores data on the commodity machines. This is the core of the hadoop framework. This also provides a very high aggregate bandwidth across the cluster.
- Hadoop YARN – This is the resource-management platform which is responsible for managing computer resources over the clusters and using them for scheduling of users' applications.

Hadoop MapReduce – This is the programming model used for large scale data processing.

The Hadoop Distributed File System or the HDFS is a distributed file system that runs on commodity hardware. It is very similar to any existing distributed file system. However, there are significant differences from other distributed file systems. HDFS is designed to be highly fault-tolerant and can be deployed on a low-cost hardware. It also provides high throughput access to application data and is suitable to handle applications that have large data sets.

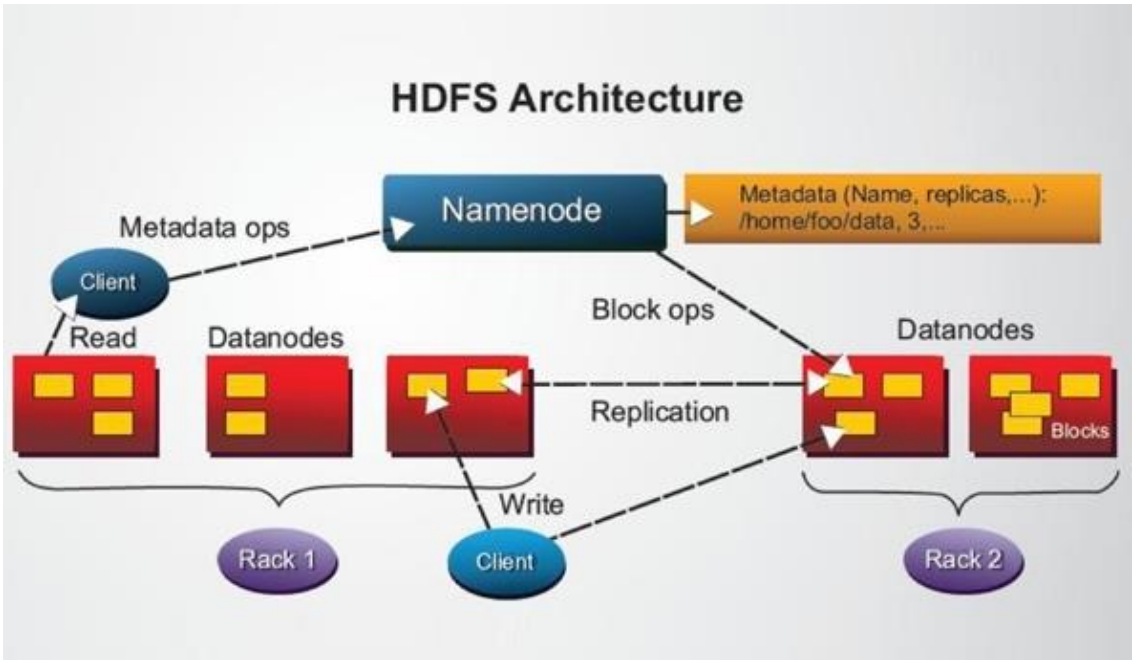


Figure 6.1- HDFS Architecture

All the components of the Hadoop ecosystem, as explicit entities are evident. The holistic view of Hadoop architecture gives prominence to Hadoop common, Hadoop YARN, Hadoop Distributed File Systems (HDFS) and Hadoop MapReduce of Hadoop Ecosystem. Hadoop common provides all java libraries, utilities, OS level abstraction, necessary java files and script to run Hadoop, while Hadoop YARN is a framework for job scheduling and cluster resource management. HDFS in Hadoop architecture provides high throughput access to application data and Hadoop MapReduce provides YARN based parallel processing of large data sets.

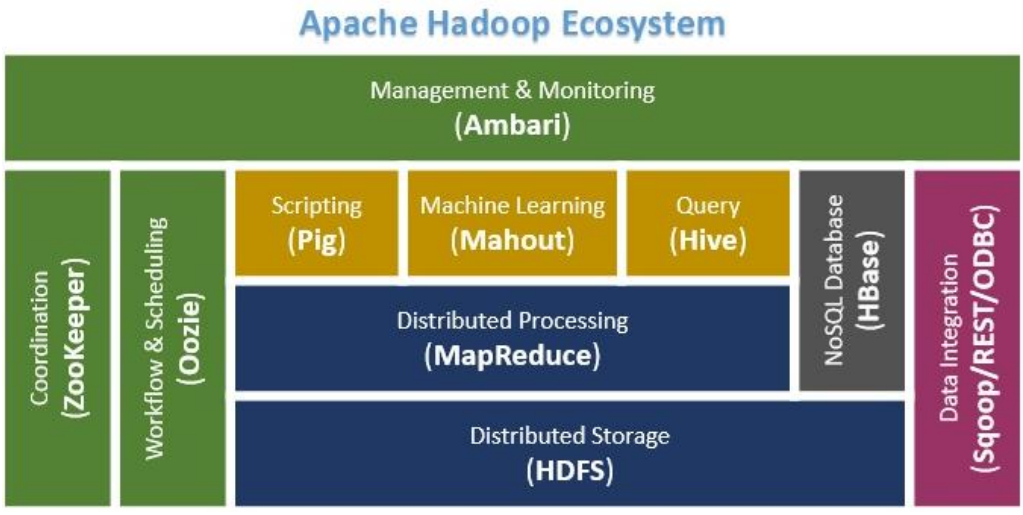


Figure 6.2 – Hadoop ecosystem

6.2 - BENEFITS OF HDFS

HDFS comes with some salient features. These features are of point of interest for many users. These are listed below –

- Hadoop distributed file system or HDFS, is a perfect match for distributed storage and distributed processing over the commodity hardware. Hadoop is fault tolerant, scalable, and very easy to scale up or down. MapReduce, which is well known for its simplicity and applicability in case of large set of distributed applications, comes as an integral part of Hadoop.
- HDFS is highly configurable. The default configuration setup is good and strong enough to support most of the applications. In general, the default configuration needs to be tuned only for very large clusters.
- Hadoop is written in Java which is supported on almost all major platforms.
- Hadoop supports shell-like commands to interact with HDFS directly.
- The NameNode and Datanodes have their own built in web servers which make it easy to check current status of the cluster.
- New features and updates are frequently implemented in HDFS. The following list is a subset of the useful features available in HDFS:
    - File permissions and authentication.
    - Rack awareness: this helps to take a node's physical location into account while scheduling tasks and allocating storage.
    - Safemode: this is the administrative mainly used mode for maintenance purpose.
    - fsck: this is a utility used to diagnose health of the file system, and to find missing files or blocks.
    - fetchdt: this is a utility used to fetch DelegationToken and store it in a file on the local system.
    - Rebalancer: this is tool used to balance the cluster when the data is unevenly distributed among DataNodes.
    - Upgrade and rollback: once the software is upgraded, it is possible to roll back to the HDFS' state before the upgrade in case of any unexpected problems.
    - Secondary NameNode: this node performs periodic checkpoints of the namespace and helps keep the size of file containing log of HDFS modifications within certain limits at the NameNode.

- Checkpoint node: this node performs periodic checkpoints of the namespace and helps minimize the size of the log stored at the NameNode containing changes to the HDFS. Replaces the role previously filled by the Secondary NameNode, though is not yet battle hardened. The NameNode allows multiple Checkpoint nodes simultaneously, as long as there are no Backup nodes registered with the system.
- Backup node: this node is an extension to the Checkpoint node. In addition to check-pointing, it also receives a stream of edits from the NameNode and maintains its own in-memory copy of the namespace, which is always in sync with the active NameNode namespace state. Only one Backup node may be registered with the NameNode at once.

High level view of HDFS.

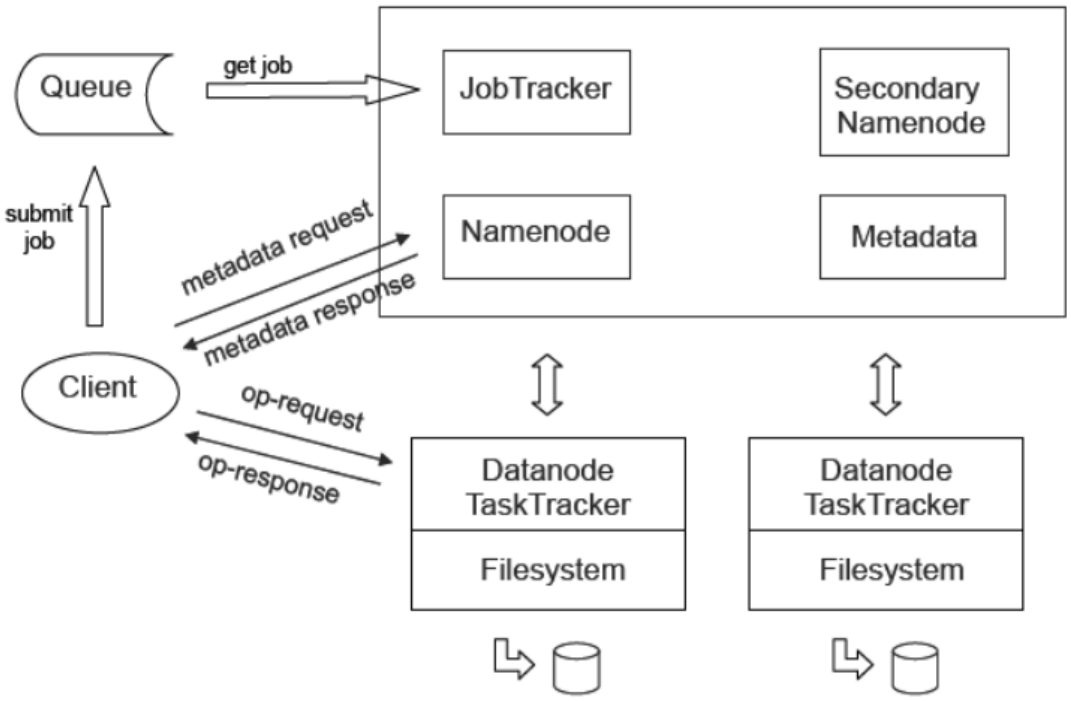


Figure 6.3- high level view of hdfs

The HDFS architecture consists of namenodes and datanodes. The call is initiated in the client component, which calls the NameNode for file metadata or file modifications. Once the name node responses, the client then takes up the task of performing the actual file I/O operation straight away with the DataNodes.

6.3 - WORKING OF NAMENODE

The HDFS namespace consists of files and directories. These files and directories are represented by inodes on the NameNode. These Inodes have the task to keep a track of attributes e.g. permissions, modification and access times, the allotted quota for namespace and disk space. Content of the file is broken into large blocks usually a size of 128 megabytes, but user can also set the block size depending upon the situation. And each block of the file is independently replicated at multiple DataNodes. Normally the data is replicated on three datanode instances but user can set this count as per need.

- NameNode is the centerpiece of HDFS.
- NameNode is also known as the Master
- NameNode only stores the metadata of HDFS – the directory tree of all files in the file system, and tracks the files across the cluster.
- NameNode does not store the actual data or the dataset. The data itself is actually stored in the DataNodes.
- NameNode knows the list of the blocks and its location for any given file in HDFS. With this information NameNode knows how to construct the file from blocks.
- NameNode is so critical to HDFS and when the NameNode is down, HDFS/Hadoop cluster is inaccessible and considered down.
- NameNode is a single point of failure in Hadoop cluster.
- NameNode is usually configured with a lot of memory (RAM). Because the block locations are help in main memory.
- MetaData consist of mapping of files to the block.
- Data is stored with datanodes.
- Name Node periodically receives the heart beat signal from the data nodes.
    - If NameNode does not receives heart beat signal, it assumes the data node is down.

NameNode asks other alive data nodes to replicate the lost blocks.

6.4 - WORKING OF DATANODE

- DataNode is responsible for storing the actual data in HDFS.

- DataNode is also known as the Slave
- NameNode and DataNode are in constant communication.
- When a DataNode starts up it announce itself to the NameNode along with the list of blocks it is responsible for.
- When a DataNode is down, it does not affect the availability of data or the cluster. NameNode will arrange for replication for the blocks managed by the DataNode that is not available.
- DataNode is usually configured with a lot of hard disk space. Because the actual data is stored in the DataNode.

TYPICAL DATA NODE CONFIGURATION:

- Processors: 2 Quad Core CPUs running @ 2 GHz
- RAM: 64 GB
- Disk: 12-24 x 1TB SATA
- Network: 10 Gigabit Ethernet

The DataNode replica block consists of two files on the local filesystem. The first file is for the data while the second file is for recording the block's metadata. The metadata here includes the checksums for the data and the generation stamp. The data file size should be the same of the actual length of the block. It and does not require any extra space to round it up to the nominal block size as in the traditional file systems. Hence if any of the blocks is half full it requires only half of the space of the full block on the local drive.

During the startup each DataNode connects to its corresponding NameNode and does the handshaking. This handshaking verifies the namespace ID and the software version of the DataNode. If there is any mismatch found, the DataNode goes down automatically.

6.4.1 - HOW A FILE IS STORED ON hdfs

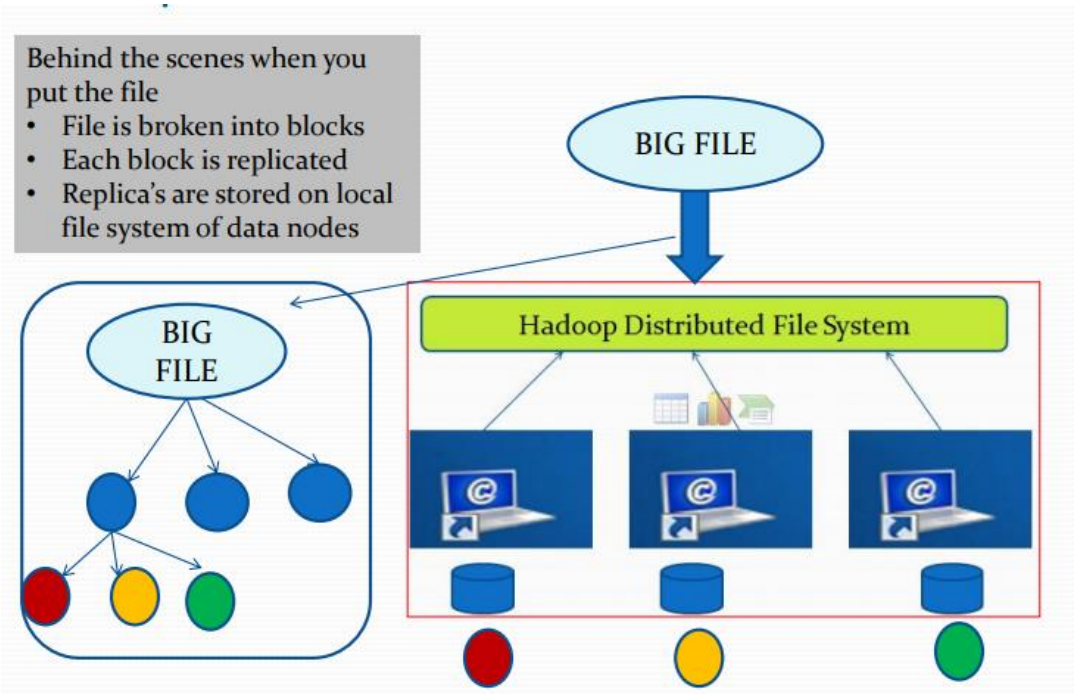


Figure 6.4 – Normal replication

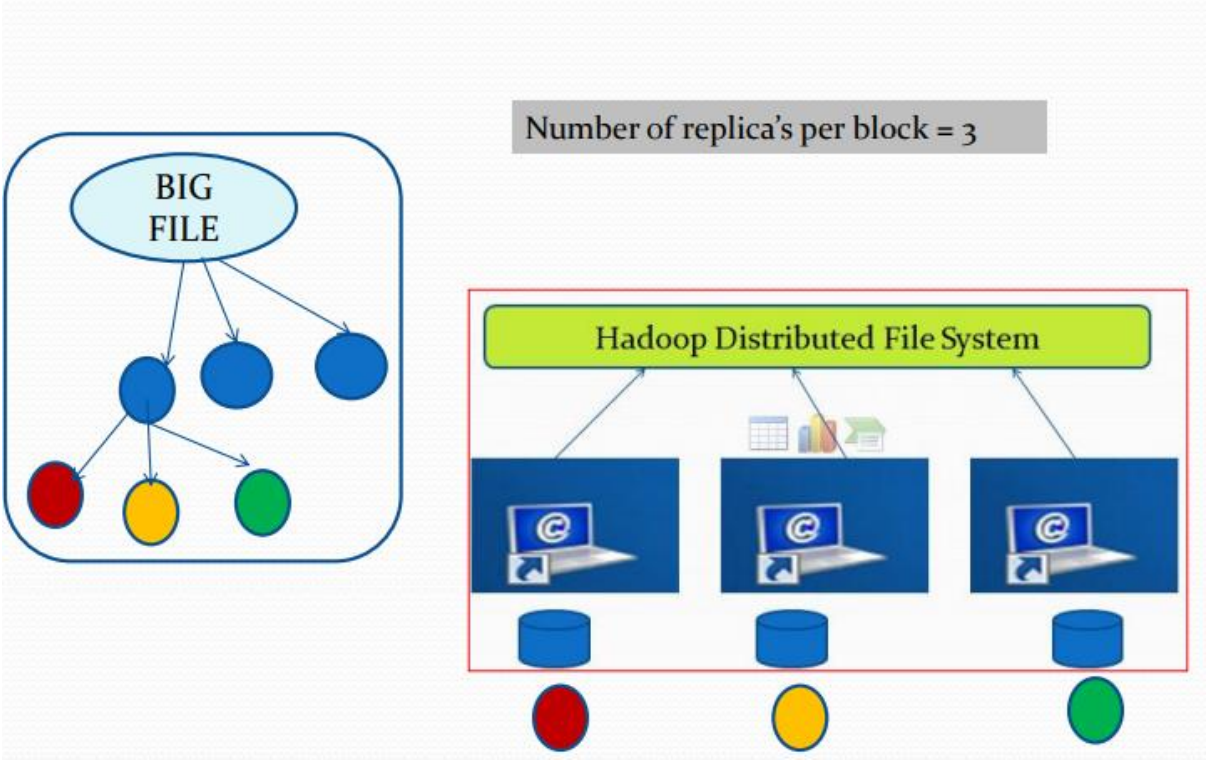


Figure 6.5 - Under replicated block.

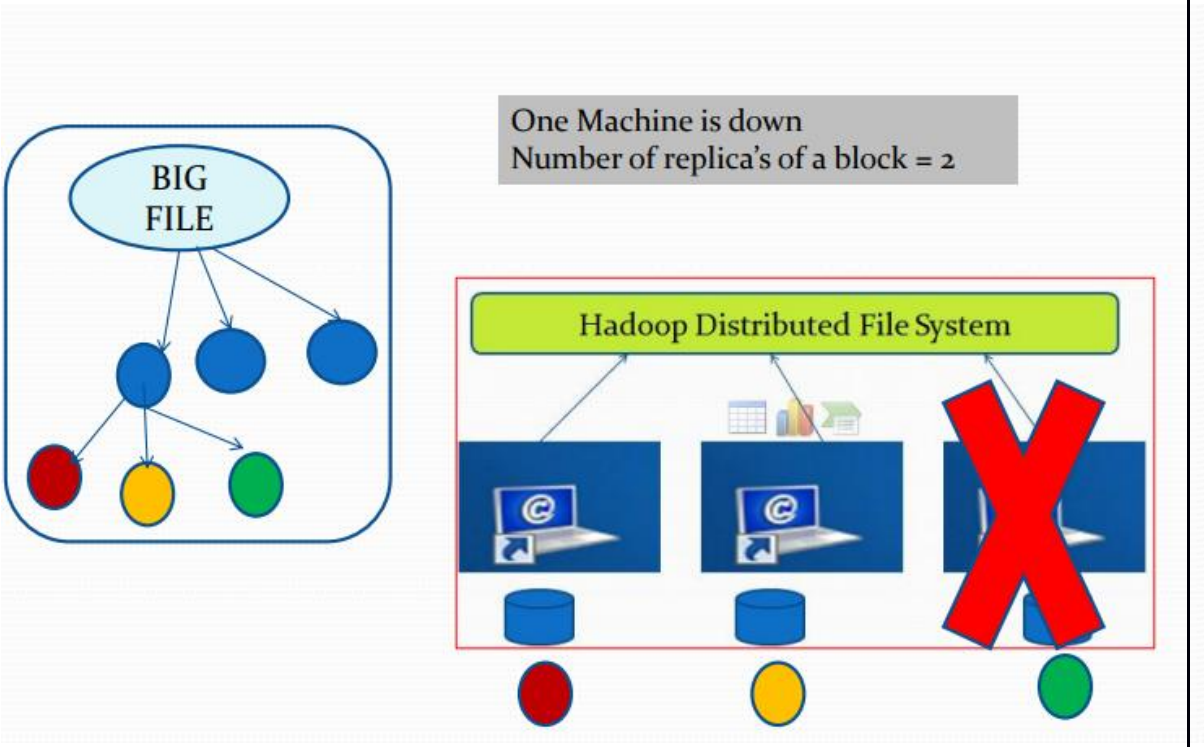


Figure 6.6 - Managing failed node.

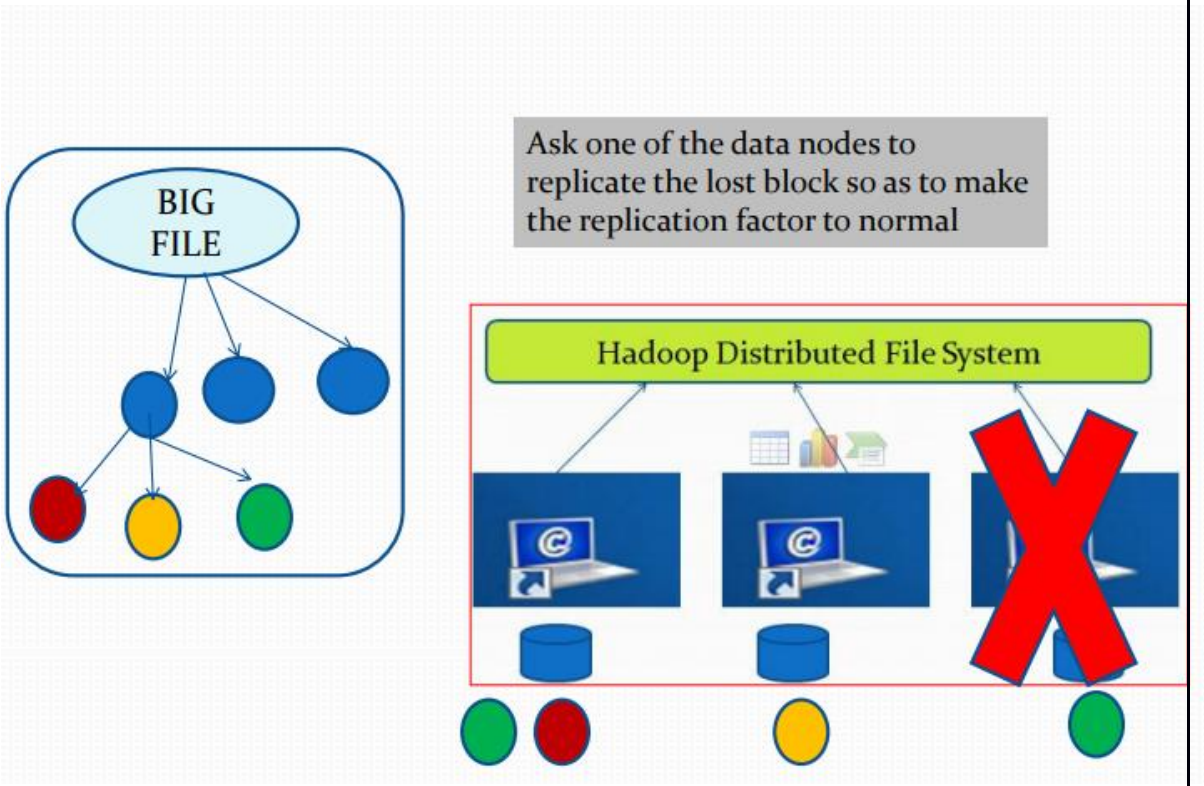


Figure 6.7 - Over replicated block.

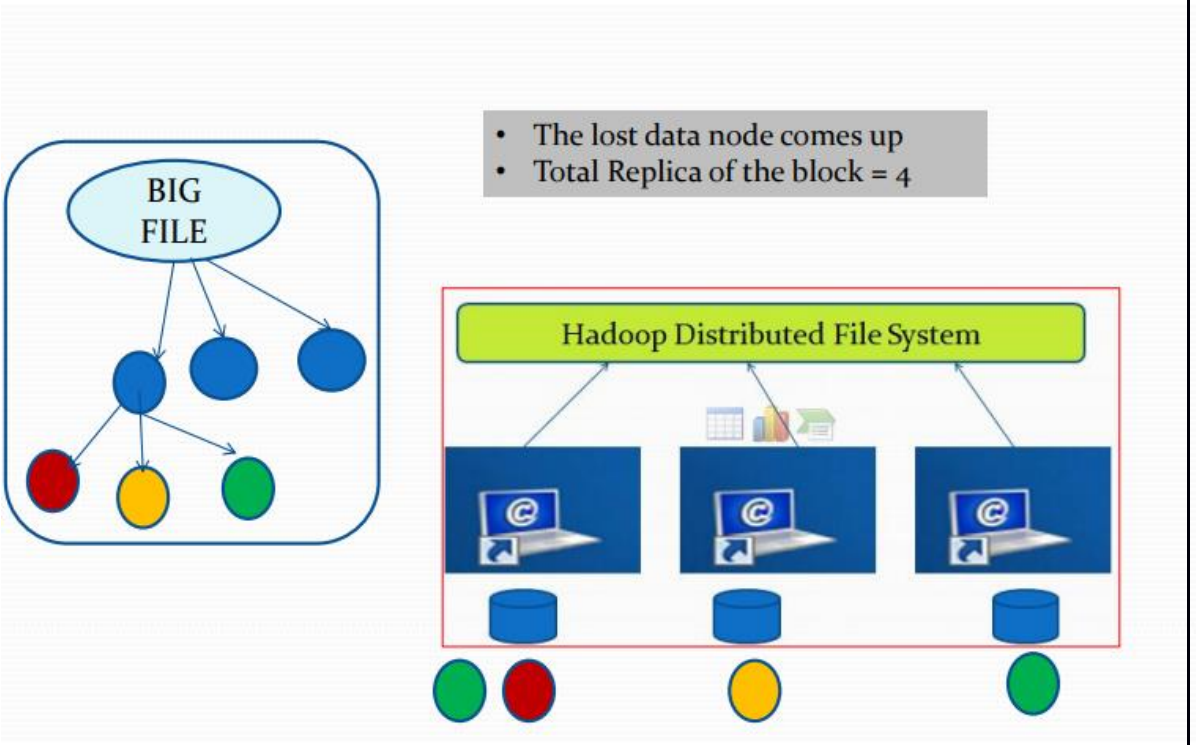
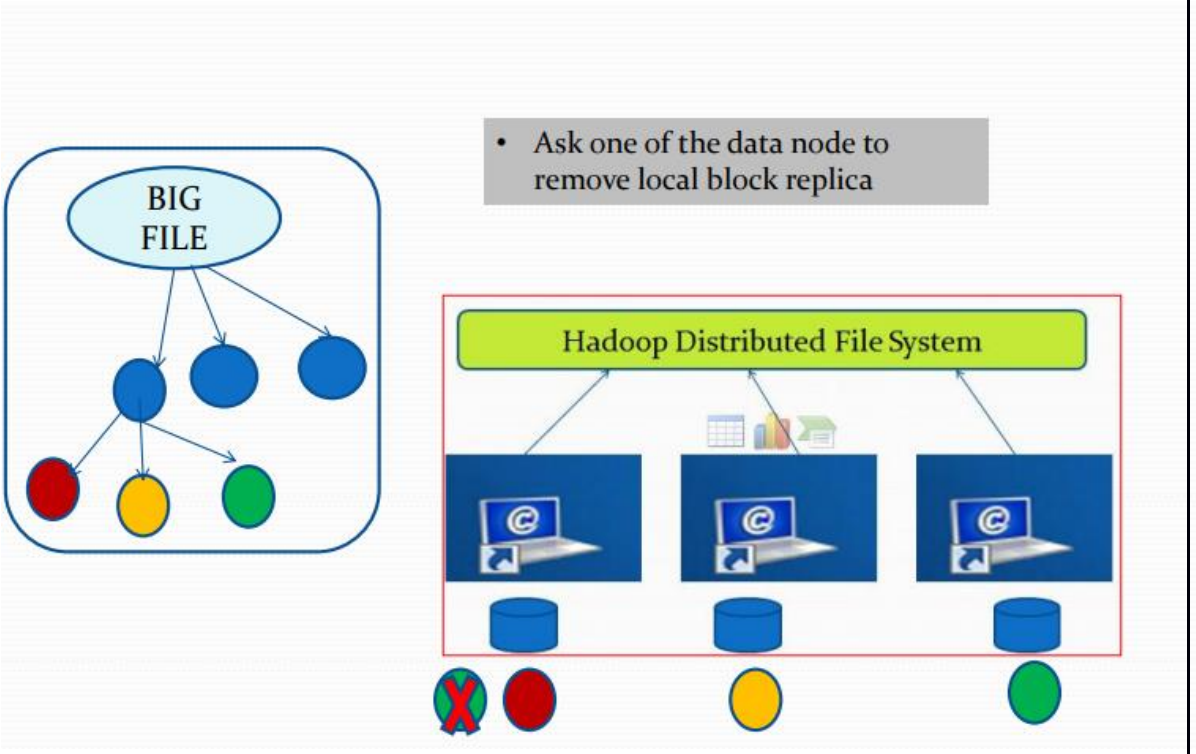


Figure 6.8 - Managing over replicated block.



- Sends heart beat signal to NameNode periodically.
  - Sends block report.
  - Storage capacity.
  - Number of data transfers happening.

- Will be considered down if not able to send the heart beat.

- NameNode never contacts DataNodes directly.
  - Replies to heart beat signal.
- Data Node receives following instructions from the name node as part of heart beat signal.
  - Replicate the blocks (In case of under replicated blocks).
  - Remove the local block replica (In case of over replicated blocks).
- NameNode make sure that replication factor is always remains normal (default is 3).

6.5 - WORKING OF SECONDARY NAMENODE

Secondary Namenode does not serve as a backup namenode but it only periodically reads file system changes log and ap-plies them to fsimage file in order to bring the system upto date. This allows the system to start faster next time.

Secondary NameNode in hadoop is a specially dedicated node in HDFS cluster whose main function is to take checkpoints of the file system metadata present on namenode. It is not a backup namenode. It just checkpoints namenode's file system namespace. The Secondary NameNode is a helper to the primary NameNode but not replace for primary namenode.

As the NameNode is the single point of failure in HDFS, if NameNode fails entire HDFS file system is lost. So in order to overcome this, Hadoop implemented Secondary NameNode whose main function is to store a copy of FsImage file and edits log file.

FsImage is a snapshot of the HDFS file system metadata at a certain point of time and EditLog is a transaction log which contains records for every change that occurs to file system metadata. So, at any point of time, applying edits log records to FsImage (recently saved copy) will give the current status of FsImage, i.e. file system metadata.

6.5.1- CHECKPOINTING

- HDFS metadata can be thought of consisting of two parts: the base filesystem table (stored in a file called **fsimage**) and the edit log which lists changes made to the base table (stored in a file called **edits**).
- Checkpointing is a process of reconciling fsimage with edits to produce a new

version of fsimage.

- There are two benefits arising out of this: a more **recent version of fsimage**, and **a truncated edit log**.
- fsimage is like a snapshot of the state of the filesystem as at particular moment whereas the edit log is a list of changes to the filesystem state since then.
- When the **namenode starts up** it reads the **fsimage file as its starting point** and then **applies any edits** from the log and then starts listening to data nodes for details of where the blocks of data reside (this information is not stored in fsimage or the edit log but rather is maintained in the memory of the name node).

Figure 6.9 - checkpointing

6.5.2 - MAIN FUCTIONS

Stores a copy of FsImage file and edits log.

- Periodically applies edits log records to FsImage file and refreshes the edits log. And sends this updated FsImage file to NameNode so that NameNode doesn't need to re-apply the EditLog records during its start up process. Thus Secondary NameNode makes NameNode start up process fast.
- If NameNode is failed, File System metadata can be recovered from the last saved FsImage on the Secondary NameNode but Secondary NameNode can't take the primary NameNode's functionality.
- Check pointing of File system metadata is performed.
- Over a period of time edits file can become very big and the next start become very longer.
- Secondary NameNode merges the edits file contents periodically with fsimage file to keep the edits file size within a sizeable limit.

6.6 - YARN

YARN forms an integral part of Hadoop 2.0.YARN is great enabler for dynamic resource utilization on Hadoop framework as users can run various Hadoop applications without having to bother about increasing workloads.

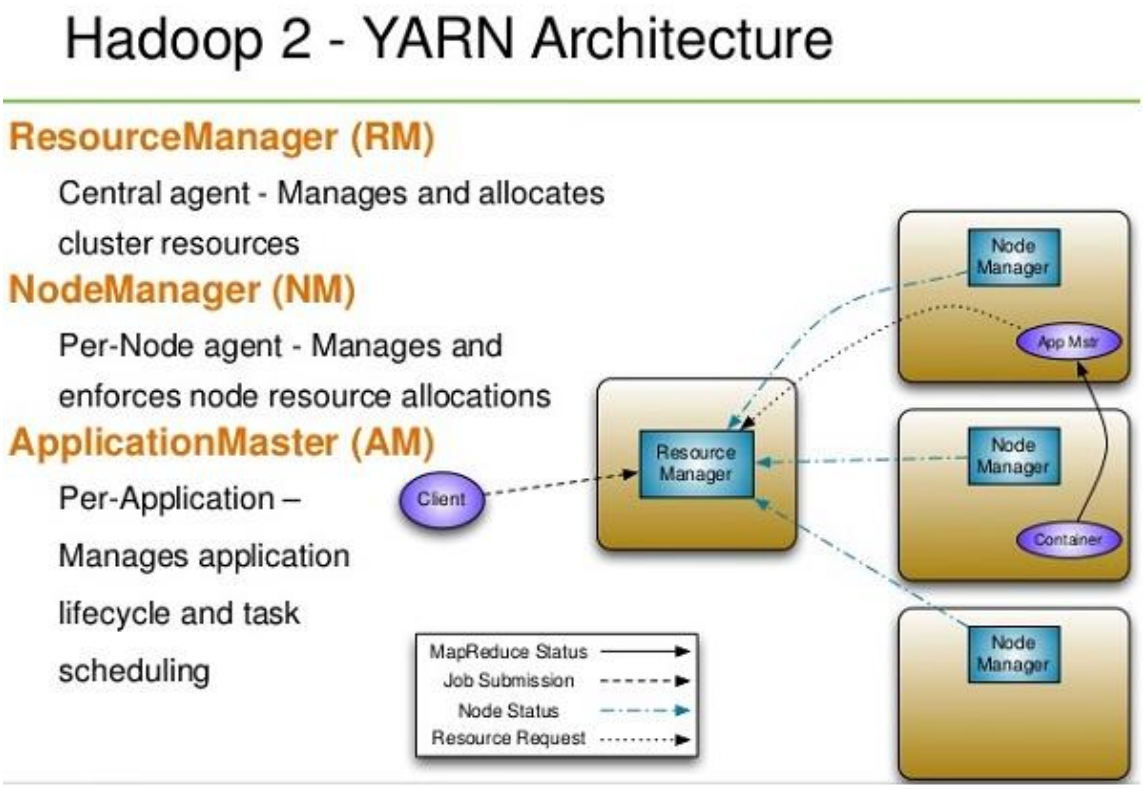


Figure 6.10 – yarn architecture

Key Benefits of Hadoop 2.0 YARN Component-

- It offers improved cluster utilization.
- Highly scalable.
- Beyond Java.
- Novel programming models and services.
- Agility.
- Stands for Yet Another Resource Negotiator and was introduced in Hadoop 2.0.
- In Yarn, the job tracker is split into two different daemons called Resource Manager and Node Manager (node specific). The resource manager only manages the allocation of resources to the different jobs apart from comprising a scheduler which just takes care of the scheduling jobs without worrying about any monitoring or status updates. Different resources such as memory, cpu time,

network bandwidth etc. are put into one unit called the Resource Container. There are different AppMasters running on different nodes which talk to a number of these resource containers and accordingly update the Node Manager with the monitoring/status details.

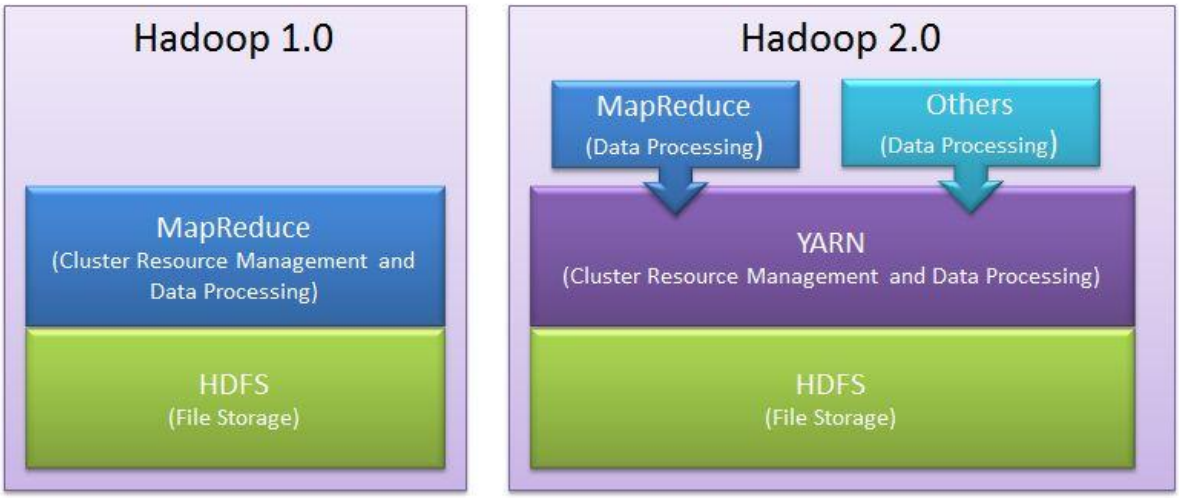


Figure 6.11 - YARN

6.6.1 - YARN USE CASE

Yahoo has close to 40,000 nodes running Apache Hadoop with 500,000 MapReduce jobs per day taking 230 compute years extra for processing every day. YARN at Yahoo helped them increase the load on the most heavily used Hadoop cluster to 125,000 jobs a day when compared to 80,000 jobs a day which is close to 50% increase.

6.6.2 - ADVANTAGES OF YARN

- **Yarn does efficient utilization of the resource.** There are no more fixed map-reduce slots. YARN provides central resource manager. With YARN, you can now run multiple applications in Hadoop, all sharing a common resource.
- **Yarn can even run application that do not follow MapReduce model.**
- YARN decouples MapReduce's resource management and scheduling capabilities from the data processing component, enabling Hadoop to support more varied processing approaches and a broader array of applications. For example, Hadoop clusters can now run interactive querying and streaming data applications simultaneously with MapReduce batch jobs. This also streamlines MapReduce to do what is does best - process data.
- **YARN is backward compatible.** This means that existing MapReduce job can

run on Hadoop 2.0 without any change.

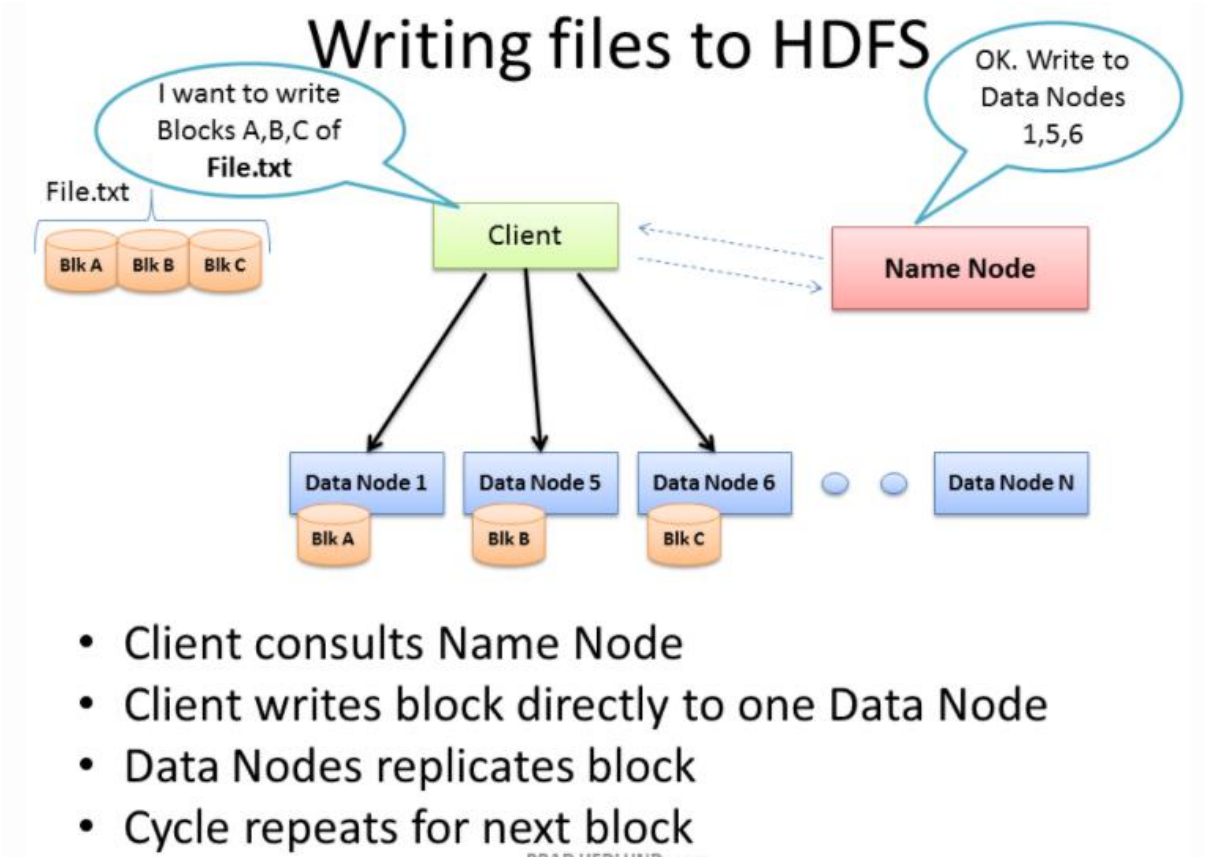
- No more JobTracker and TaskTracker needed in Hadoop 2.0
- **JobTracker and TaskTracker has totally disappeared**. YARN splits the two major functionalities of the JobTracker i.e. resource management and job scheduling/monitoring into 2 separate daemons (components).
  - Resource Manager.
  - Node Manager(node specific).

6.7 - WRITING A FILE ON HDFS

Hadoop cluster is useless until it has data. Suppose a huge file named 'File.txt' is to be loaded into the cluster for processing. The goal here is fast parallel processing of lots of data. To accomplish that we need as many machines as possible working on this data all at once. To that end, the Client is going to break the data file into smaller "Blocks", and place those blocks on different machines throughout the cluster. The more blocks, the more machines that will be able to work on this data in parallel. At the same time, these machines may be prone to failure, so we want to insure that every block of data is on multiple machines at once to avoid data loss. So each block will be replicated in the cluster as its loaded. The standard setting for Hadoop is to have (3) copies of each block in the cluster. This can be configured with the dfs.replication parameter in the file hdfs-site.xml.

The Client breaks File.txt into (3) Blocks. For each block, the Client consults the Name Node (usually TCP 9000) and receives a list of (3) Data Nodes that should have a copy of this block. The Client then writes the block directly to the Data Node (usually TCP 50010). The receiving Data Node replicates the block to other Data Nodes, and the cycle repeats for the remaining blocks. The Name Node is not in the data path. The Name Node only provides the map of where data is and where data should go in the cluster (file system metadata).

Figure 6.12 writing files on HDFS



6.7.1- PREPARING HDFS WRITE

The Client is ready to load File.txt into the cluster and breaks it up into blocks, starting with Block A. The Client consults the Name Node that it wants to write File.txt, gets permission from the Name Node, and receives a list of (3) Data Nodes for each block, a unique list for each block. The Name Node used its Rack Awareness data to influence the decision of which Data Nodes to provide in these lists. The key rule is that for every block of data, two copies will exist in one rack, another copy in a different rack. So the list provided to the Client will follow this rule.

Before the Client writes "Block A" of File.txt to the cluster it wants to know that all Data Nodes which are expected to have a copy of this block are ready to receive it. It picks the first Data Node in the list for Block A (Data Node 1), opens a TCP 50010 connection and says, "Hey, get ready to receive a block, and here's a list of (2) Data Nodes, Data Node 5 and Data Node 6. Go make sure they're ready to receive this block too." Data Node 1 then opens a TCP connection to Data Node 5 and says, "Hey, get ready to receive a block, and go make sure Data Node 6 is ready is receive this block too." Data Node 5 will then ask Data Node 6, "Hey, are you ready to receive a block?"

The acknowledgments of readiness come back on the same TCP pipeline, until the initial Data Node 1 sends a "Ready" message back to the Client. At this point the Client is ready to begin writing block data into the cluster.

6.7.2 - PIPELINE WRITE

- As data for each block is written into the cluster a replication pipeline is created between the (3) Data Nodes (or however many you have configured in dfs.replication). This means that as a Data Node is receiving block data it will at the same time push a copy of that data to the next Node in the pipeline.
- Here too is a primary example of leveraging the Rack Awareness data in the Name Node to improve cluster performance. Notice that the second and third Data Nodes in the pipeline are in the same rack, and therefore the final leg of the pipeline does not need to traverse between racks and instead benefits from in-rack bandwidth and low latency. The next block will not be begin until this block is successfully written to all three nodes.

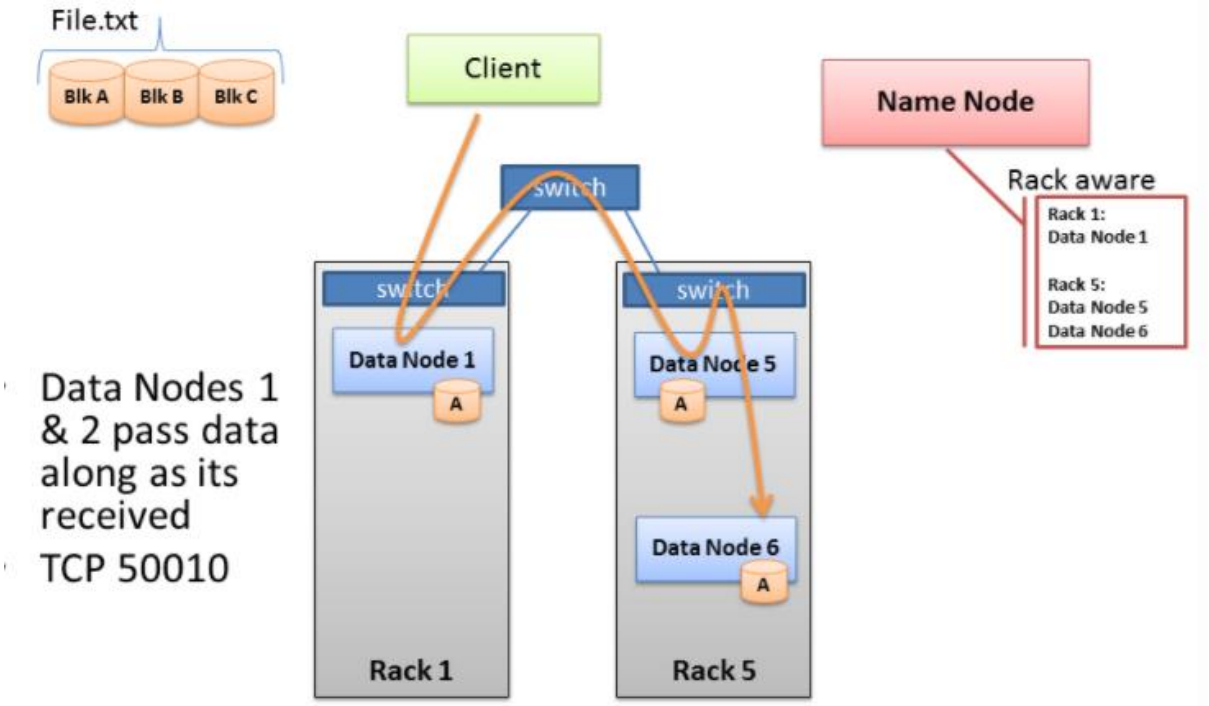


Figure 6.13 – file writing process on HDFS

When all three Nodes have successfully received the block they will send a "Block Received" report to the Name Node. They will also send "Success" messages back up the pipeline and close down the TCP sessions. The Client receives a success message and tells the Name Node the block was successfully written. The Name Node updates

it metadata info with the Node locations of Block A in File.txt. The Client is ready to start the pipeline process again for the next block of data.

6.8 - **READING A FILE FROM HDFS**

- Connects to NameNode.
- Ask NameNode to give the list of data nodes that is hosting the replica's of the block of file.
- Client then directly read from the data nodes without contacting again to NameNode.
- Along with the data, check sum is also shipped for verifying the data integrity
  - If the replica is corrupt client intimates NameNode, and try to get the data from other DataNode.
- There are some cases in which a Data Node daemon itself will need to read a block of data from HDFS. One such case is where the Data Node has been asked to process data that it does not have locally, and therefore it must retrieve the data from another Data Node over the network before it can begin processing.

This is another key example of the Name Node's Rack Awareness knowledge providing optimal network behavior. When the Data Node asks the Name Node for location of block data, the Name Node will check if another Data Node in the same rack has the data. If so, the Name Node provides the in-rack location from which to retrieve the data. The flow does not need to traverse two more switches and congested links find the data in another rack. With the data retrieved quicker in-rack, the data processing can begin sooner, and the job completes that much faster.

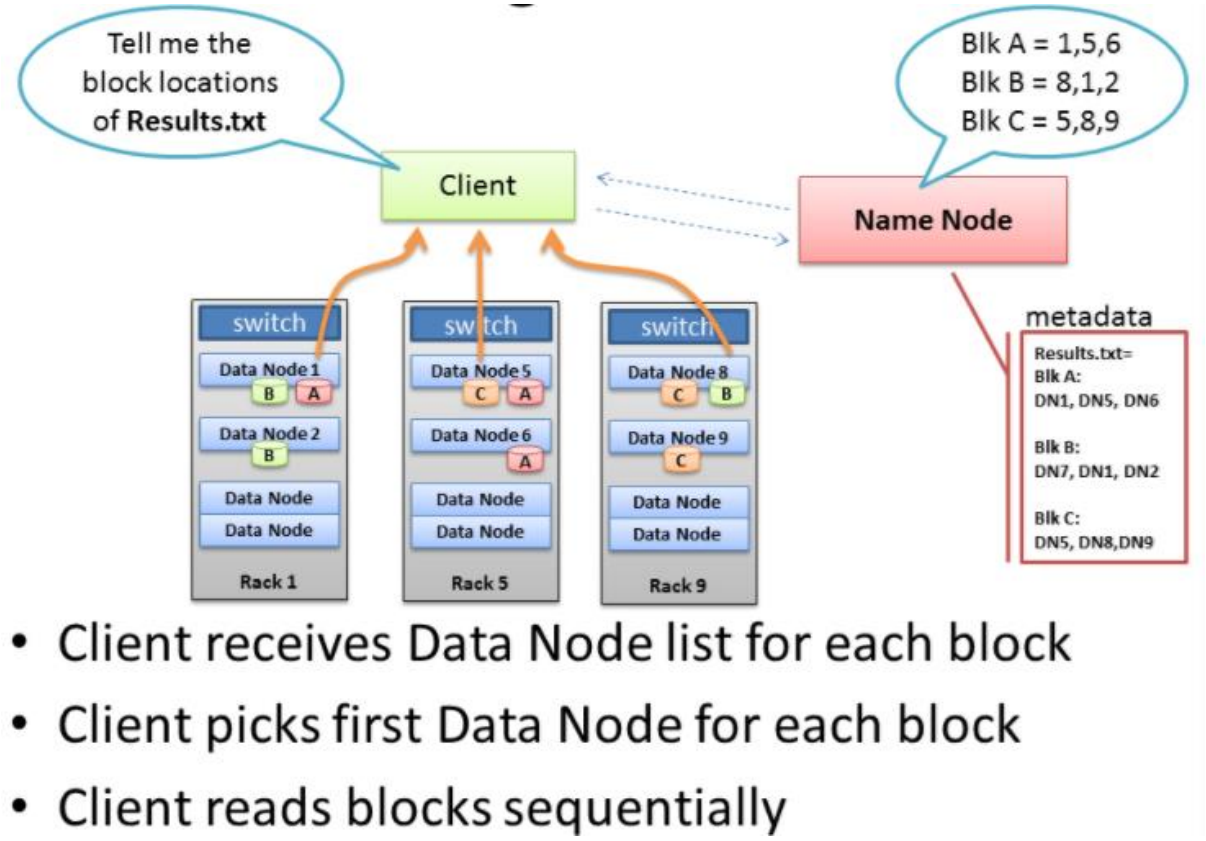


Figure 6.14 - Client reading file

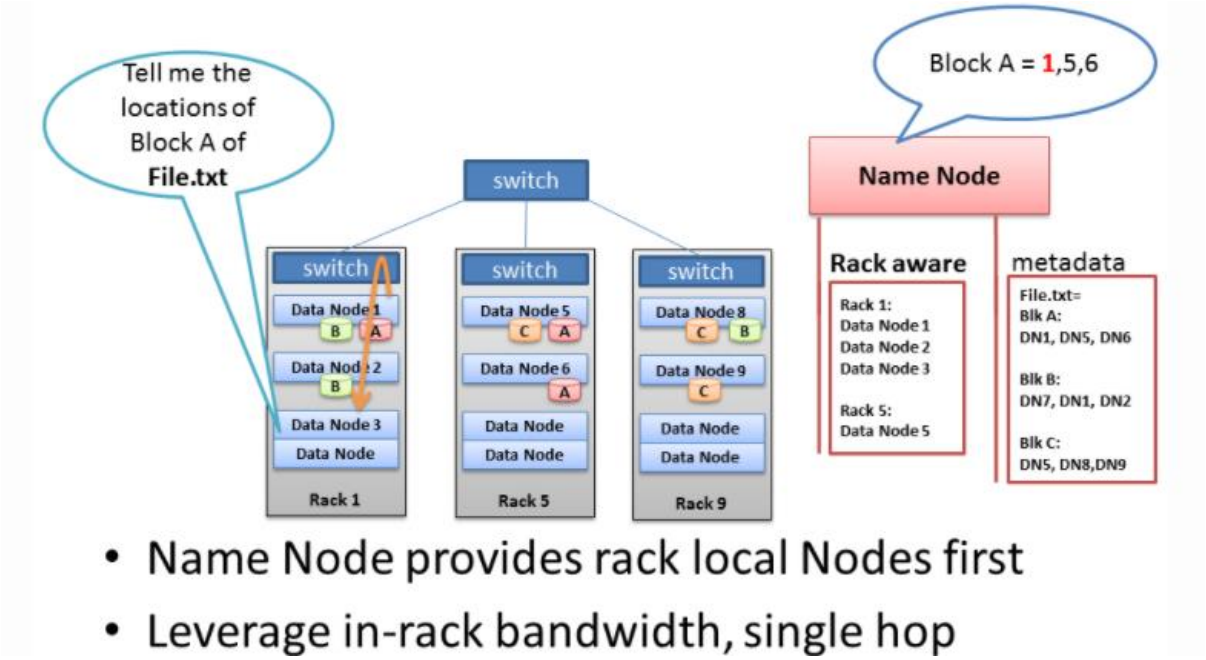


Figure 6.15 DataNode reading file.

CHAPTER 7 MAPREDUCE FRAMEWORK BASICS

An Overview

- Philosophy of map reduce.
- What is MapReduce job?
- Mapping Phase.
- Reduce Phase.
- Data flow with zero, single and multiple Reducers
- MapReduce job demo

7.1 - Philosophy of Map-Reduce

MapReduce™ is the heart of Apache™ Hadoop®. It is this programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. The MapReduce concept is fairly simple to understand for those who are familiar with clustered scale-out data processing solutions.

For people new to this topic, it can be somewhat difficult to grasp, because it's not typically something people have been exposed to previously. If you're new to Hadoop's MapReduce jobs, don't worry: we're going to describe it in a way that gets you up to speed quickly.

The term MapReduce actually refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

Map and Reduce

Hadoop MapReduce (Hadoop Map/Reduce) is a software framework for distributed processing of large data sets on compute clusters of commodity hardware. It is a sub-project of the Apache Hadoop project. The framework takes care of scheduling tasks, monitoring them and re-executing any failed tasks.

According to The Apache Software Foundation, the primary objective of Map/Reduce is to split the input data set into independent chunks that are processed in a completely parallel manner. The Hadoop MapReduce framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically, both the input and the output of the job are stored in a file system.

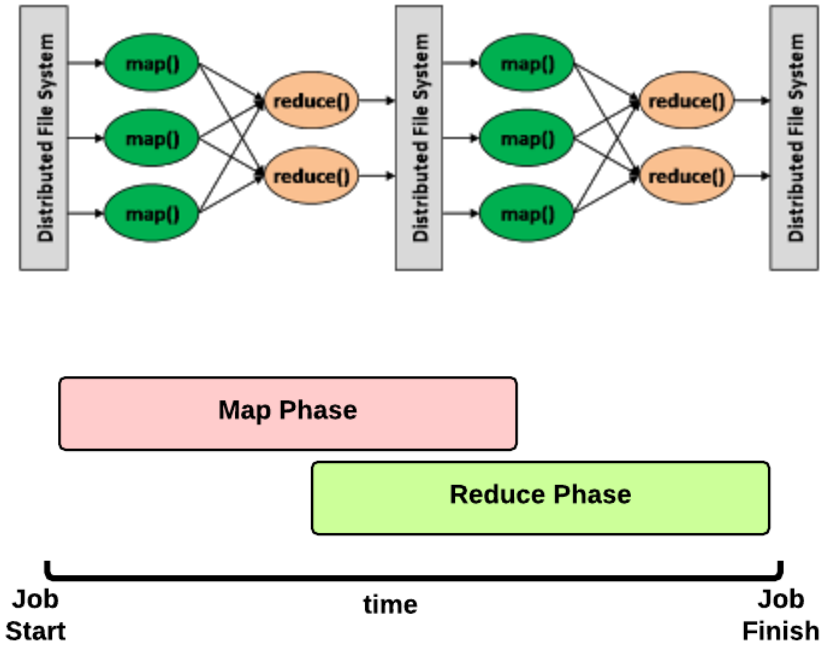


Figure 7.1 MapReduce Logical Data flow

The diagram is a Unix pipeline, which mimics the whole MapReduce flow.

"Map" step: Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of redundant input data is processed.

"Shuffle" step: Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker ode.

"Reduce" step: Worker nodes now process each group of output data, per key, in parallel.

The model is a specialization of the split-apply-combine strategy for data analysis.[7.1]

7.2 - What is MapReduce job?

The distributed file system is designed to handle large files (multi-GB) with sequential read/write operation. Each file is broken into chunks, and stored across multiple data nodes as local OS files.

There is a master “NameNode” to keep track of overall file directory structure and the placement of chunks. This NameNode is the central control point and may re-distributed replicas as needed. DataNode reports all its chunks to the NameNode at

bootup. Each chunk has a version number which will be increased for all update. Therefore, the NameNode know if any of the chunks of a DataNode is stale (e.g. when the DataNode crash for some period of time). Those stale chunks will be garbage collected at a later time.

To read a file, the client API will calculate the chunk index based on the offset of the file pointer and make a request to the NameNode. The NameNode will reply which DataNodes has a copy of that chunk. From this points, the client contacts the DataNode directly without going through the NameNode.

To write a file, client API will first contact the NameNode who will designate one of the replica as the primary (by granting it a lease). The response of the NameNode contains who is the primary and who are the secondary replicas. Then the client push its changes to all DataNodes in any order, but this change is stored in a buffer of each DataNode. After changes are buffered at all DataNodes, the client send a "commit" request to the primary, which determines an order to update and then push this order to all other secondaries. After all secondaries complete the commit, the primary will response to the client about the success. All changes of chunk distribution and metadata changes will be written to an operation log file at the NameNode. This log file maintain an order list of operation which is important for the NameNode to recover its view after a crash. The NameNode also maintain its persistent state by regularly check-pointing to a file. In case of the NameNode crash, a new NameNode will take over after restoring the state from the last checkpoint file and replay the operation log.

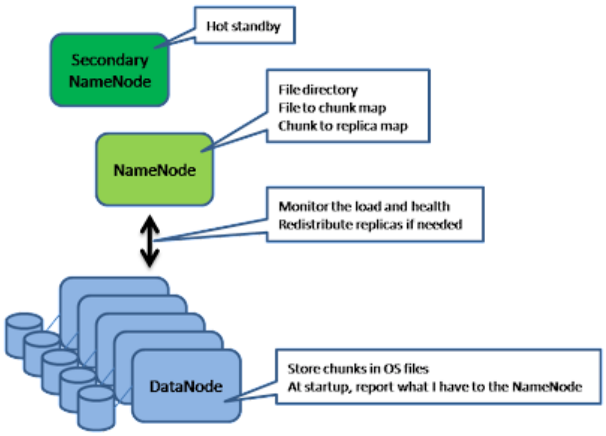


Figure 7.2 HDFS

MAP-Reduce job:

The job execution starts when the client program submit to the JobTracker a job configuration, which specifies the map, combine and reduce function, as well as the input and output path of data.

The JobTracker will first determine the number of splits (each split is configurable, ~16-64MB) from the input path, and select some TaskTracker based on their network proximity to the data sources, then the JobTracker send the task requests to those selected TaskTrackers.

Each TaskTracker will start the map phase processing by extracting the input data from the splits. For each record parsed by the "InputFormat", it invoke the user provided "map" function, which emits a number of key/value pair in the memory buffer. A periodic wakeup process will sort the memory buffer into different reducer node by invoke the "combine" function. The key/value pairs are sorted into one of the R local files (suppose there are R reducer nodes).

When the map task completes (all splits are done), the TaskTracker will notify the JobTracker. When all the TaskTrackers are done, the JobTracker will notify the selected TaskTrackers for the reduce phase.

Each TaskTracker will read the region files remotely. It sorts the key/value pairs and for each key, it invoke the "reduce" function, which collects the key/aggregatedValue into the output file (one per reducer node).

Map/Reduce framework is resilient to crash of any components. The JobTracker keep tracks of the progress of each phases and periodically ping the TaskTracker for their health status. When any of the map phase TaskTracker crashes, the JobTracker will reassign the map task to a different TaskTracker node, which will rerun all the assigned splits. If the reduce phase TaskTracker crashes, the JobTracker will rerun the reduce at a different TaskTracker.

After both phase completes, the JobTracker will unblock the client program.

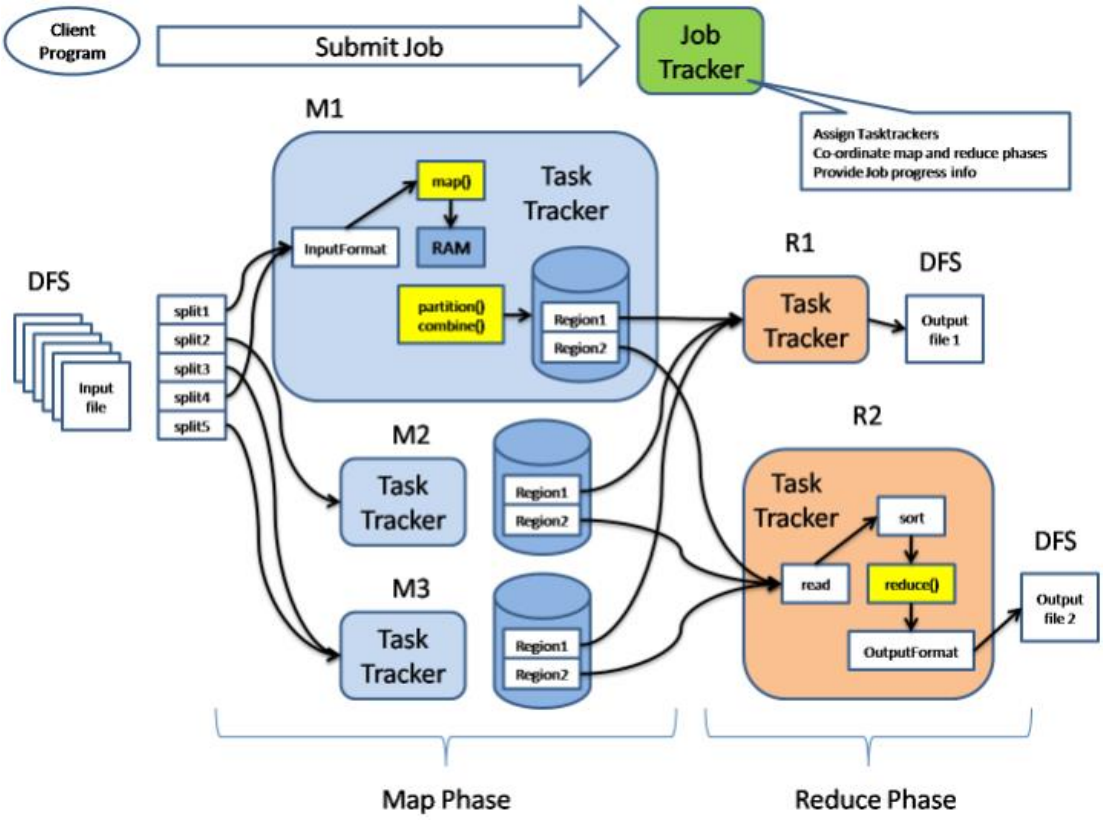


Figure 7.3 – Map-reduce Mechanism.

7.3 - Mapping Phase

A MapReduce application processes the data in input splits on a record-by-recordbasis and that each record is understood by MapReduce to be a key/value pair. After the input splits have been calculated, the mapper tasks can start processing them — that is, right after the Resource Manager's scheduling facility assigns them their processing resources. (In Hadoop 1, the JobTracker assigns mapper tasks to specific processing slots.)

The mapper task itself processes its input split one record at a time — in the figure, this lone record is represented by the key/value pair . In the case of our flight data, when the input splits are calculated (using the default file processing method for text files), the assumption is that each row in the text file is a single record.

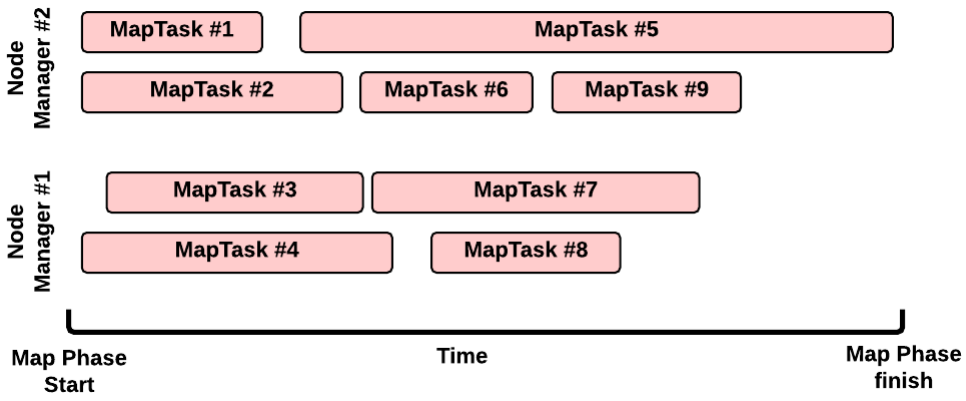


Figure 7.4 - Mapping

Mapper can use or can ignore the input key (in_key).

And it can emit:

- Zero key/value pair.
- One key/value pair.
- "n" key/value pair.

Map(k1,v1) → list(k2,v2)

Map function is called for one record at a time.

Input Split consist of records. For each record in the input split, map function will be called. Each record will be sent as key –value pair to map function. So map functions knows only one thing, the current record. Neither does it keep the state of whether how many records it has processed or how many records will appear.

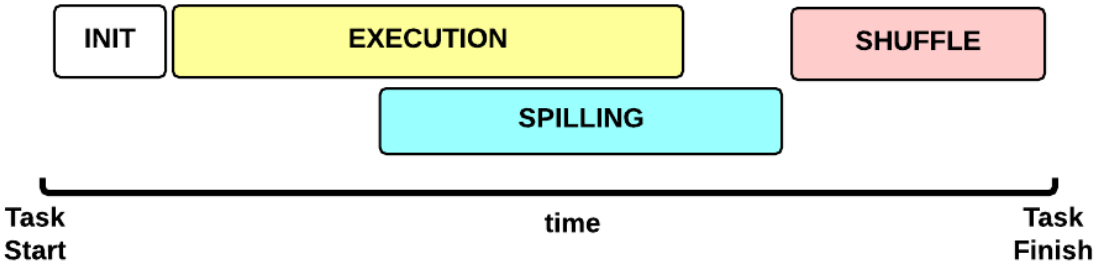


Figure 7.5 Map task execution timeline

7.4 - Reduce Phase

This phase after the mapping phase is completed. In this phase there is a combination of the Shufflestage and the Reduce stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

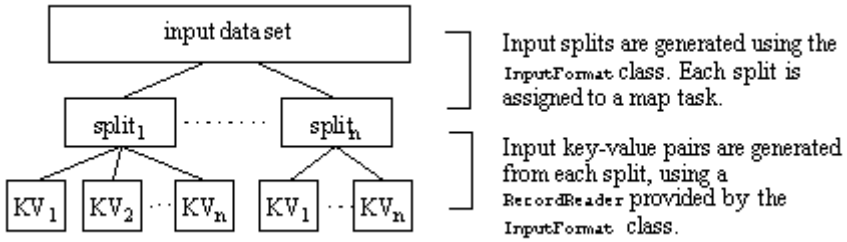


Figure 7.6 – Map reduce overview.

7.5 - Data flow with zero, single and multiple Reducers

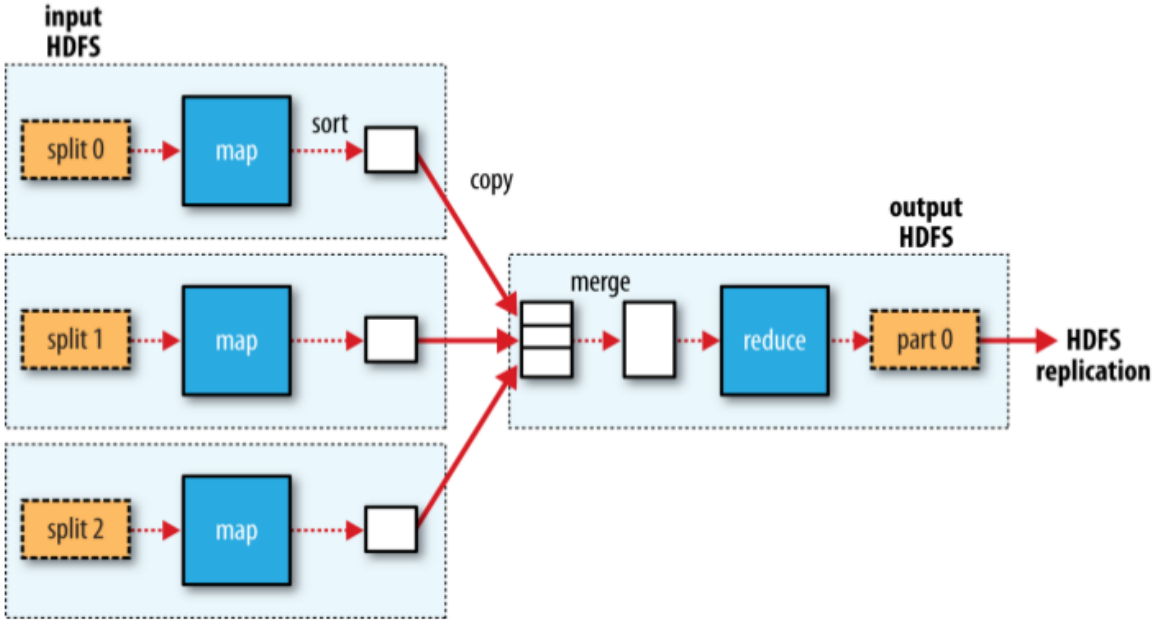


Figure 7.7 - MapReduce Data flow with single reduce task

Reduce tasks don't have the advantage of data locality; the input to a single reduce task is normally the output from all mappers. In the present example, we have a single reduce task that is fed by all of the map tasks. Therefore, the sorted map outputs have to be transferred across the network to the node where the reduce task is running, where they are merged and then passed to the user-defined reduce function. Writing the reduce output does consume network bandwidth, but only as much as a normal HDFS write pipeline consumes.

Solid arrow shows the data transfer between the nodes.

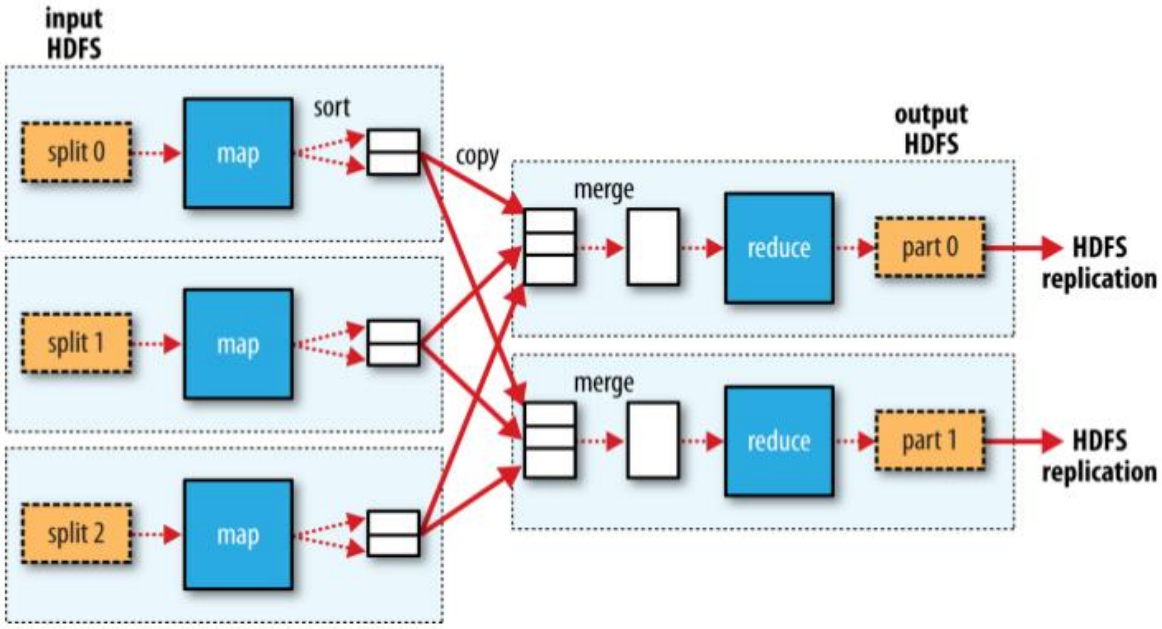


Figure 7.8 MapReduce Data flow with multiple reduce task

The number of reduce tasks is not governed by the size of the input, but instead is specified independently.

When there are multiple reducers, the map tasks partition their output, each creating one partition for each reduce task. There can be many keys (and their associated values) in each partition, but the records for any given key are all in a single partition. The partitioning can be controlled by a user-defined partitioning function, but normally the default partitioner—which buckets keys using a hash function—works very well.

This diagram makes it clear why the data flow between map and reduce tasks is colloquially known as "the shuffle," as each reduce task is fed by many map tasks. The shuffle is more complicated than this diagram.

Finally, it's also possible to have zero reduce tasks. This can be appropriate when you don't need the shuffle because the processing can be carried out entirely in parallel. In this case, the only off-node data transfer is when the map tasks write to HDFS.

7.6 - MapReduce job demo

We already saw to mimic MapReduce job using Unix pipe. Let's us see how well it performs when simulating and when run as a true MapReduce job in Hadoop:

MapReduce as Unix pipe

The data we will use is from the National Climatic Data Center or NCDC. It is stored using a line-oriented ASCII format, in which each line is a record. The format supports a rich set of meteorological elements, many of which are optional or with variable data lengths. For simplicity, we focus on the basic elements, such as temperature, which is always present and is of fixed width.

To simulate MapReduce as Unix Pipe, we would need three things:

1. Mapper
2. Reducer
3. NCDC Data

For the sake of simplicity, the we will analyze the weather data during the years 1949 and 1950. The sample data looks like this:

0067011990999991**950**051507004999999N9+**00651**+99999999999

0043011990999991**950**051512004999999N9+**00461**+99999999999

0043011990999991**950**051518004999999N9-**00711**+99999999999

0043012650999991**949**032412004050000 1N9+**00921**+99999999999

0043012650999991**949**032418004050000 1N9+**01191**+99999999999…

It contains 1 million records from around 10,000 weather stations.

For the Mapper and Reducer, we can user any language like **JAVA, Ruby, C, C#,** but for my program, I have used **Python v3[7.2]**.

Our goal is to:

1. Get Data
2. Forward it to mapper
3. Shuffle the output of mapper
4. Feed to Reducer
5. Display to standard output.

-To get the data, we will use “cat” command.

-Using pipe, we forward data to Map function.

-“sort” command will shuffle the data.

-Pipe to reduce function.

-At last, use redirection to output to a file (omit for standard output).

Therefore, the skeleton command will be:

% cat <in_file> | map.py | sort | reduce > <out_file>

PROGRAM:

Mapper:

```
#!/usr/bin/env python3.6
import re, sys
for line in sys.stdin:
      val = line.strip()
      (year, temp, q) = (val[15:19], val[37:42], val[42:43])
      if (temp != "+9999" and re.match("[01459]", q)):
            print("%s\t%s" % (year, temp))
```

Reducers:

```
#!/usr/bin/env python3.6
import sys
last_key = None
max_val = -sys.maxsize
for line in sys.stdin:
      key, val = line.strip().split("\t")
      if last_key and last_key != key:
            print("%s\t%s" % (last_key, max_val))
            (last_key, max_val) = (key, int(val))
      else:
            (last_key, max_val) = (key, max(max_val, int(val)))
if last_key:
      print("%s\t%s" % (last_key, max_val))
```

%cat Sim_map_red/ncdc_49-50 | Sim_map_red/map.py | sort | Sim_map_red/reduce.py

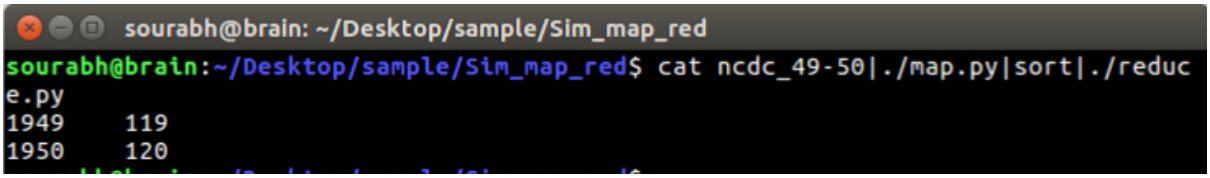


Figure 7.9 MapReduce using Unix Pipe

Hadoop provides an API to MapReduce that allows you to write your map and reduce functions in languages other than Java. Hadoop Streaming uses Unix standard streams as the interface between Hadoop and your program, so you can use any language that can read standard input and write to standard output to write your MapReduce.

Streaming is naturally suited for text processing. Map input data is passed over standard input to your map function, which processes it line by line and writes lines to standard output. A map output key-value pair is written as a single tab-delimited line. Input to the reduce function is in the same format—a tab-separated key-value pair—passed over standard input. The reduce function reads lines from standard input, which the framework guarantees are sorted by key, and writes its results to standard output.

It is discussed in detail in the Advanced MapReduce chapter.

CHAPTER 8 MAP REDUCE FRAMEWORK PART - 2

An Overview

- Working of Map-Reduce work.
- How input split is processed by the mapper.
- Overall process.
- Working.
- Performance consideration.
- Uses.

8.1 - WORKING OF MAP-REDUCE WORK

MapReduce is a programming model suitable for processing of huge data. Hadoop is capable of running MapReduce programs written in various languages: Java, Ruby, Python, and C++. MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.[1][2]

MAP REDUCE PROGRAMS CONSISTS OF TWO PHASES :

- Map phase.
- Reduce phase.

Input to each phase are key-value pairs. In addition, every programmer needs to specify two functions: map function and reduce function.

The whole process goes through three phase of execution namely.

8.2 - HOW INPUT IS SPLIT IS PROCESSED BY THE MAPPER

- Input split by default is the block size (dfs.block.size).
- Each input split / block comprises of records
  - A record is one line in the input split terminated by "\n" (new line character).
- Every input format has "RecordReader".
  - RecordReader reads the records from the input split.
  - RecordReader reads ONE record at a time and call the map function.
    - If the input split has 4 records, 4 times map function will be called, one for each record

It sends the record to the map function in key value pair.

8.2.1 - FLOW OF WORD COUNT PROBLEM

Demonstration of word count problem:

Consider the have following input data for the MapReduce Program:

Welcome to Hadoop Class

Hadoop is good

Hadoop is bad

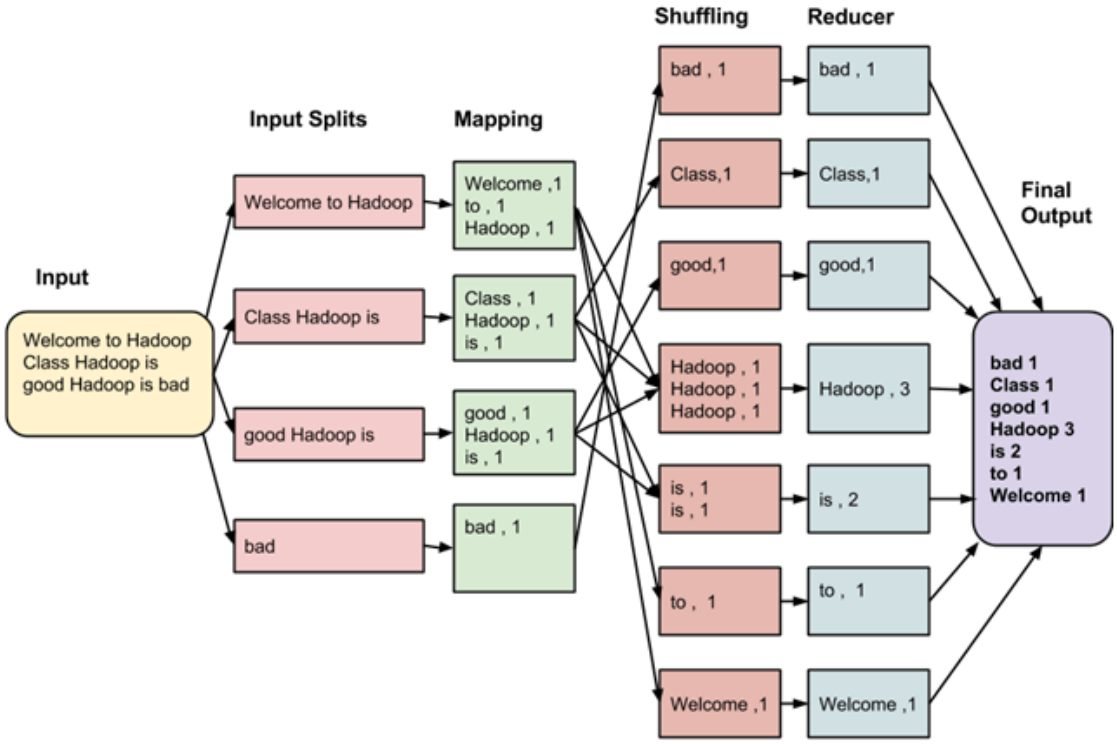


Figure 8.1 – map reduce word count

The final output looks like:

bad	1
Class	1
good	1
Hadoop	3
is	2
to	1
Welcome	1

Figure 8.2 – word count

The model is a specialization of the split-apply-combine strategy for data analysis[3].

The data goes through three phases:

8.2.2 - INPUT SPLITS

Input to a MapReduce job is divided into fixed-size pieces called input splits Input split is a chunk of the input that is consumed by a single map.

8.2.3 - MAPPING

This is very first phase in the execution of map-reduce program. In this phase data in each split is passed to a mapping function to produce output values. In our example, job of mapping phase is to count number of occurrences of each word from input splits (more details about input-split is given below) and prepare a list in the form of <word, frequency>

8.2.4 - SHUFFLING

This phase consumes output of Mapping phase. Its task is to consolidate the relevant records from Mapping phase output. In our example, same words are clubed together along with their respective frequency.

8.2.5 - REDUCING

In this phase, output values from Shuffling phase are aggregated. This phase combines values from Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.

In our example, this phase aggregates the values from Shuffling phase i.e., calculates total occurrences of each words.

8.3 - OVERALL PROCESS

- One map task is created for each split which then executes map function for each record in the split.
- It is always beneficial to have multiple splits, because time taken to process a split is small as compared to the time taken for processing of the whole input. When the splits are smaller, the processing is better load balanced since we are processing the splits in parallel.
- However, it is also not desirable to have splits too small in size. When splits are too small, the overload of managing the splits and map task creation begins to dominate the total job execution time.
- For most jobs, it is better to make split size equal to the size of an HDFS block (which is 64 MB, by default).

- Execution of map tasks results into writing output to a local disk on the respective node and not to HDFS.
- Reason for choosing local disk over HDFS is, to avoid replication which takes place in case of HDFS store operation.
- Map output is intermediate output which is processed by reduce tasks to produce the final output.
- Once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication becomes overkill.
- In the event of node failure before the map output is consumed by the reduce task, Hadoop reruns the map task on another node and re-creates the map output.
- Reduce task don't work on the concept of data locality. Output of every map task is fed to the reduce task. Map output is transferred to the machine where reduce task is running.
- On this machine the output is merged and then passed to the user defined reduce function.
- Unlike to the map output, reduce output is stored in HDFS (the first replica is stored on the local node and other replicas are stored on off-rack nodes). So, writing the reduce output

8.4 - WORKING

Hadoop divides the job into tasks. There are two types of tasks:

- Map tasks (Spilts & Mapping)
- Reduce tasks (Shuffling, Reducing)as mentioned above.

The complete execution process (execution of Map and Reduce tasks, both) is controlled by two types of entities called:

- Jobtracker : Acts like a master (responsible for complete execution of submitted job)
- Multiple Task Trackers : Acts like slaves, each of them performing the job

For every job submitted for execution in the system, there is one Jobtracker that resides on Namenode and there are multiple tasktrackers which reside on Datanode.

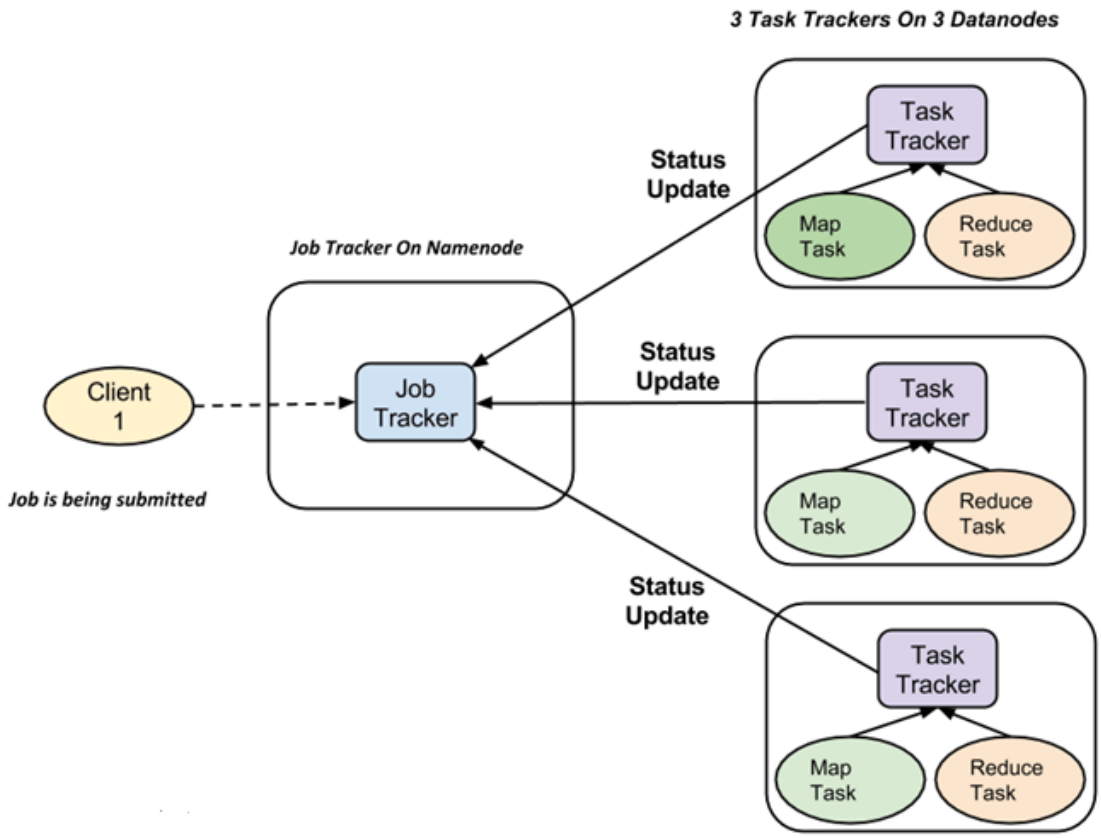


Figure 8.3 – job tracker

- A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.
- It is the responsibility of jobtracker to coordinate the activity by scheduling tasks to run on different data nodes.
- Execution of individual task is then look after by tasktracker, which resides on every data node executing part of the job.
- Tasktracker's responsibility is to send the progress report to the jobtracker.
- In addition, tasktracker periodically sends 'heartbeat' signal to the Jobtracker so as to notify him of current state of the system.
- Thus jobtracker keeps track of overall progress of each job. In the event of task failure, the jobtracker can reschedule it on a different tasktracker.

8.5 - PERFORMANCE CONSIDERATION

MapReduce programs are not guaranteed to be fast. The main benefit of this programming model is to exploit the optimized shuffle operation of the platform, and only having to write the Map and Reduce parts of the program. In practice, the author

of a MapReduce program however has to take the shuffle step into consideration; in particular the partition function and the amount of data written by the Map function can have a large impact on the performance and scalability. Additional modules such as the Combiner function can help to reduce the amount of data written to disk, and transmitted over the network. MapReduce applications can achieve sub-linear speedups under specific circumstances.[4]

In tuning performance of MapReduce, the complexity of mapping, shuffle, sorting (grouping by the key), and reducing has to be taken into account. The amount of data produced by the mappers is a key parameter that shifts the bulk of the computation cost between mapping and reducing. Reducing includes sorting (grouping of the keys) which has nonlinear complexity. Hence, small partition sizes reduce sorting time, but there is a trade-off because having a large number of reducers may be impractical. The influence of split unit size is marginal (unless chosen particularly badly, say <1MB). The gains from some mappers reading load from local disks, on average, is minor.[5]

8.6 - USES

MapReduce is useful in a wide range of applications, including distributed pattern-based searching, distributed sorting, web link-graph reversal, Singular Value Decomposition,[6]web access log stats, inverted index construction, document clustering, machine learning,[7] and statistical machine translation. Moreover, the MapReduce model has been adapted to several computing environments like multi-core and many-core systems, desktop grids, multi-cluster,[8] volunteer computing environments, dynamic cloud environments, mobile environments, and high-performance computing environments.

At Google, MapReduce was used to completely regenerate Google's index of the World Wide Web. It replaced the old ad hoc programs that updated the index and ran the various analyses, Development at Google has since moved on to technologies such as Percolator, FlumeJava and MillWheel that offer streaming operation and updates instead of batch processing, to allow integrating "live" search results without rebuilding the complete index.[9]

MapReduce's stable inputs and outputs are usually stored in a distributed file system. The transient data are usually stored on local disk and fetched remotely.

CHAPTER 9 HADOOP STREAMING

An Overview

- JOB 1
- JOB 2

9.1 - INTRODUCTION

Hadoop provides an API to MapReduce that allows you to write your map and reduce functions in languages other than Java. Hadoop Streaming uses Unix standard streams as the interface between Hadoop and your program, so you can use any language that can read standard input and write to standard output to write your MapReduce program.

Streaming is naturally suited for text processing (although, as of version 0.21.0, it can handle binary streams, too), and when used in text mode, it has a line-oriented view of data. Map input data is passed over standard input to your map function, which processes it line by line and writes lines to standard output. A map output key-value pair is written as a single tab-delimited line. Input to the reduce function is in the same format—a tab-separated key-value pair—passed over standard input. The reduce func-

tion reads lines from standard input, which the framework guarantees are sorted by key, and writes its results to standard output.

Basically streaming supports any language that can read from standard input, and write to standard output. So streaming jobs using python is also possible.

For this following steps are followed:(make sute that hadoop services are up and running check using ‘JPS’ command)

1 : Locate ‘hadoop streaming .jar file’ for streaming jobs in hadoop. For this ‘grep’ command is used.

- ‘locate hadoop-streaming | grep jar’

2 : The jar location found in this case is ‘/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.2.jar’ (it may be different depending upon your installation)

```
sourabh@brain:~$ jps
2881 DataNode
3409 NodeManager
3281 ResourceManager
3107 SecondaryNameNode
3749 Jps
2749 NameNode
sourabh@brain:~$ locate hadoop-streaming | grep jar
/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.2.jar
/usr/local/hadoop/share/hadoop/tools/sources/hadoop-streaming-2.7.2-sources.jar
/usr/local/hadoop/share/hadoop/tools/sources/hadoop-streaming-2.7.2-test-sources
.jar
/usr/local/pig-0.17.0/test/e2e/pig/lib/hadoop-streaming.jar
sourabh@brain:~$ _
```

Figure 9.1 – locating hadoop streaming jar

3 : The streaming command consist of five parts

a) jar location

b)mapper file

c)reducer file

d)input location

e)output location

4 : Move the file into the hdfs and run the streaming command from the local directory which contains the ‘mapper’ and ‘reducer’ scripts.

Here I have moved the data-set to the HDFS directory named project.

It can be seen in the screen shot attached below.

```
sourabh@brain:~/Documents/DATA$ hadoop fs -mkdir /project
sourabh@brain:~/Documents/DATA$ hadoop fs -ls /
Found 16 items
drwxr-xr-x   - sourabh supergroup          0 2017-06-28 10:46 /4300
drwxr-xr-x   - sourabh supergroup          0 2017-06-25 22:16 /accessout
drwxr-xr-x   - sourabh supergroup          0 2017-07-20 12:31 /flumedir
drwxr-xr-x   - sourabh supergroup          0 2017-06-25 22:35 /iphitout
drwxr-xr-x   - sourabh supergroup          0 2017-06-25 12:10 /jobout
drwxr-xr-x   - sourabh supergroup          0 2017-06-25 12:31 /jobout1
drwxr-xr-x   - sourabh supergroup          0 2017-06-25 17:29 /jobout2
drwxr-xr-x   - sourabh supergroup          0 2017-06-25 19:20 /jobout3
drwxr-xr-x   - sourabh supergroup          0 2017-06-20 10:27 /mapre
drwxr-xr-x   - sourabh supergroup          0 2017-06-26 00:46 /murl
drwxr-xr-x   - sourabh supergroup          0 2017-10-21 16:19 /new
drwxr-xr-x   - sourabh supergroup          0 2017-06-20 16:09 /o_wrd_cnt
drwxr-xr-x   - sourabh supergroup          0 2017-06-20 10:49 /outmapre
drwxr-xr-x   - sourabh supergroup          0 2017-11-17 20:23 /project
drwx------   - sourabh supergroup          0 2017-07-16 16:58 /tmp
drwxr-xr-x   - sourabh supergroup          0 2017-07-10 10:41 /user
```

Figure 9.2 – making directory in HDFS for JOB

9.2- DESCRIBTION OF DATA

The data set used is called ‘Discussion board data’ or ‘forum data’. It is formally described as a hierarchical or tree-like in structure: a forum can contain a number of subforums, each of which may have several topics. Within a forum's topic, each new discussion started is called a thread, and can be replied to by as many people as so wish.

The data set is of online learning portal named ‘udacity’.

The file is named as ‘forum.node.tsv’.

The data set contains 17 columns which are:

- Id
- Title
- Tagnames
- author_id
- Body
- node_type
- parent_id
- abs_parent_id
- added_at
- score
- state_string
- last_edited_id
- last_activity_id
- last_activity_at
- activity_revision_id
- extra
- extra_ret_id

First few rows of the data are shown below.

Standard	Standard
author_id	body
100000458	
100005361	We are looking for feedback on the audio in our videos
100001178	I am sorry if I am being a nob ... but I do not seem t
100003292	When will unit 2 be online?
100003268	Hi there! ↵ Any Hungarians doing the course? We c
100003192	Please tell about the Course Application. How to use t
100008254	Is there a way to change what is and isn't publicly di

Standard	Standard	Standard	Standard	S
node_type	parent_id	abs_parent_id	added_at	s
question	\N	\N	2012-02-25 08:09:06.787181+00	1
question	\N	\N	2012-02-23 00:28:02.321344+00	2
question	\N	\N	2012-02-23 09:15:02.270861+00	0
question	\N	\N	2012-02-26 15:47:12.522262+00	0
question	\N	\N	2012-02-27 15:09:11.184434+00	0
question	\N	\N	2012-03-08 08:34:06.704674+00	-
question	\N	\N	2012-02-23 22:00:39.134386+00	0

score	state_string	last_edited_id	last_activity_by_id	last_acti
1		\N	100000921	2012-02-2
2		\N	201398145	2014-01-1
0	(closed)	10843	100001178	2012-02-2
0	(closed)	51919	100003292	2012-03-0
0		\N	100003268	2012-02-2
-1		\N	100003192	2012-03-0
0	(closed)	90176	100002517	2012-03-0

Standard	Standard	Standard	Standard
last_activity_at	active_revision_id	extra	extra_re
2012-02-25 08:11:01.623548+00	6922	\N	\N
2014-01-14 17:18:35.613939+00	2960	\N	\N
2012-02-23 10:36:43.165119+00	3513	\N	\N
2012-03-03 10:12:27.41521+00	21196	\N	\N
2012-02-27 15:09:11.184434+00	9322	\N	\N
2012-03-08 08:34:06.704674+00	34477	\N	\N
2012-03-08 22:12:34.271971+00	4876	\N	\N

	Standard	Standard	Standard	Standard	Standard
	active_revision_id	extra	extra_ref_id	extra_count	marked
+00	6922	\N	\N	204	f
+00	2960	\N	\N	524	f
+00	3513	\N	\N	169	t
00	21196	\N	\N	186	t
+00	9322	\N	\N	106	f
+00	34477	\N	\N	73	f
+00	4876	\N	\N	228	t

Figure 9.3 forum node.tsv file

10.3 JOB 1

The point of interest for the first streaming job is to find what are the tags which are mostly used on the discussion board.

This is will help the client or any concerned authority to know what are the most widely discussed topic and thus it can analysed.

SCRIPTS:

MAPPER – mapper1.py

```
#!/usr/bin/python
"""
Top Tags
This script is for seeing what are the top tags that are used in
   posts.
this is a mapper script
This returns after reading data tags seprated along with their count.

"""

import sys
import csv

def mapper(stdin):
    """ MapReduce Mapper. """
    reader = csv.reader(stdin, delimiter='\t')
    # Skip header.
    reader.next()
    writer = \
           csv.writer(
               sys.stdout, delimiter='\t', quotechar='"',
   quoting=csv.QUOTE_ALL)
```

```
    for line in reader:
        if len(line) == 19:
            node_type = line[5]
            if node_type == "question":
                tagnames_str = line[2]
                tagnames = tagnames_str.split()
                for tagname in tagnames:
                    writer.writerow([tagname, "1"])

if __name__ == "__main__":
    mapper(sys.stdin)
```

REDUCER – reducer1.py

```
#!/usr/bin/python
"""
Top Tags
This is the reducer script for doing computation on
output of mapper script.
it also displays the top 10 tags that where used in discussion board.

"""
import sys
import csv

def reducer():
    """
    MapReduce Reducer.  Has output:
        Tag Counts
    """
    reader = csv.reader(sys.stdin, delimiter='\t')
    writer = \
        csv.writer(
            sys.stdout, delimiter='\t', quotechar='"',
            quoting=csv.QUOTE_MINIMAL)
    tag_count = 0
    current_tag = None
    top_10 = []
    for line in reader:
        if len(line) == 2:
            tag = line[0]
```

```
            if current_tag is None or tag != current_tag:
                if not current_tag is None:
                    top_10 = add_record(current_tag, tag_count,
   top_10)
                tag_count = 0
                current_tag = tag

            tag_count += 1
    top_10 = add_record(current_tag, tag_count, top_10)
    for tag, tag_count in top_10:
        writer.writerow([tag, tag_count])

def add_record(tag, tag_count, top_10):
    """ Adds an item to the top 10 (if it makes it into the current
   top 10.) """
    if len(top_10) < 10 or tag_count > top_10[9][1]:
        top_10.append([tag, tag_count])
        top_10.sort(key=lambda tup: tup[1], reverse=True)
        top_10 = top_10[:10]
    return top_10

if __name__ == "__main__":
    reducer()
```

streaming command:

```
hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-
2.7.2.jar -mapper mapper1.py -reducer reducer1.py -file mapper1.py -
file reducer1.py -input /project -output /toptags
```

the output is saved on hdfs with directory named ‘toptags’

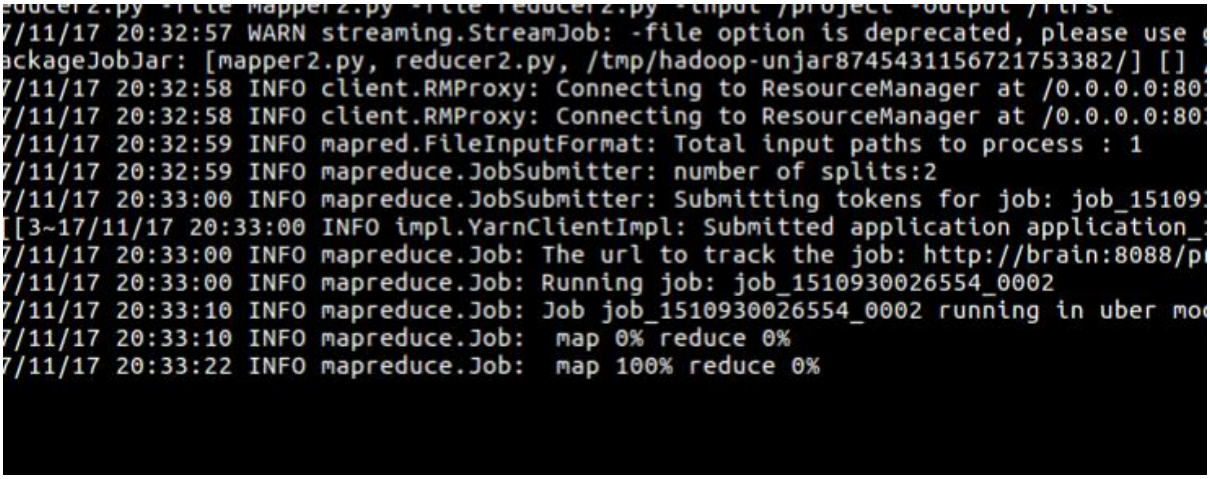


Figure 9.4 streaming job1 running

OUTPUT:

After the complete execution of the streaming command the output can be viewed by the following command:

- $ hadoop fs -cat /toptags/part-00000

OR

- Hadoop local server i.e. localhost://50070
- To move the result on the local file system we can pipe the output to a output file .

$ hadoop fs -cat /toptags/part-00000 > outputfile

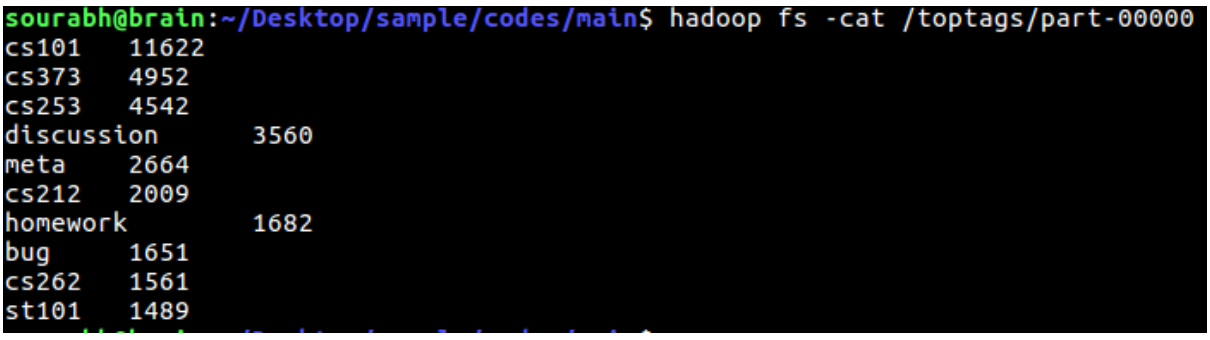


Figure 9.5 verifying successful execution of job 1

By looking at the output it can deduced that the courseware name ‘cse101’ was the most discussed topic and it came up 11,622 times on the discussion board.

```
cs101   11622
cs373   4952
cs253   4542
discussion      3560
meta    2664
cs212   2009
homework        1682
bug     1651
cs262   1561
st101   1489
```

10.4 JOB 2

In this job we are required to find what is the most active hour for a particular user or student.

This in turn gives insight at what time most of the user are active.

The outpur of this streaming job is tab delimited file that contain author id and the time when it was most active.

The job was run on the same data-set as JOB 1.

Below are the mapper and reducer script for the job.

MAPPER SCRIPT – mapper2.py

```
#!/usr/bin/python

import sys

import csv


def mapper(stdin):

    """ MapReduce Mapper. """

    reader = csv.reader(stdin, delimiter='\t')

    # Skip header.

    reader.next()


    for line in reader:

        if len(line) == 19:

            author_id = line[3]
```

```
            added_at = line[8]
            # Sample added_at:
            # 2012-02-25 08:09:06.787181+00
            added_at = added_at.strip()
            hour = added_at[11:13]

            yield '%s\t%s\t%s' % (author_id, hour, 1)

if __name__ == "__main__":
    for output in mapper(sys.stdin):
        print output
```

REDUCER SCRIPT – reducer2.py

```
#!/usr/bin/python
#student times file
import sys
import csv

def reducer():
    """ MapReduce Reducer. """
    reader = csv.reader(sys.stdin, delimiter='\t')
    writer = \
        csv.writer(
            sys.stdout, delimiter='\t', quotechar='"',
            quoting=csv.QUOTE_MINIMAL)
    current_author_id = None
    hour_counts = [0] * 24

    for line in reader:
```

```
        if len(line) == 3:
            author_id = line[0]
            hour = int(line[1])
            count = int(line[2])
            if current_author_id is None or author_id !=
current_author_id:
                if not current_author_id is None:
                    write_record(current_author_id, hour_counts,
writer)
                hour_counts = [0] * 24
                current_author_id = author_id
            hour_counts[hour] += count
    write_record(current_author_id, hour_counts, writer)


def get_max_hour(hour_counts):
    """ Returns the max hour(s). """
    max_hours = []
    max_hour_count = -1
    for i in range(24):
        if hour_counts[i] > max_hour_count:
            max_hour_count = hour_counts[i]
    for i in range(24):
        if hour_counts[i] == max_hour_count:
            max_hours.append(i)


    return max_hours


def write_record(author_id, hour_counts, writer):
    """
    Writes record in format.
        Student ID |    Hour
```

```
    """
    max_hours = get_max_hour(hour_counts)
    for max_hour in max_hours:
        new_line = [author_id, max_hour]
        writer.writerow(new_line)


if __name__ == "__main__":
    reducer()
```

STREAMING COMMAND:

```
$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.2.jar -mapper mapper2.py -reducer reducer2.py -file mapper2.py -file reducer2.py -input /project -output /sourabh2
```

THE RESULT IS STORED ON HDFS WITH DIRECTORY NAMED 'SOURABH2'

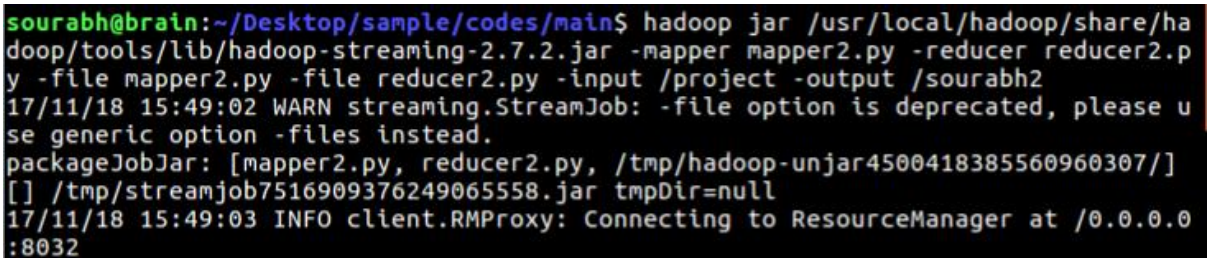


Figure 9.6 streaming job 2 started

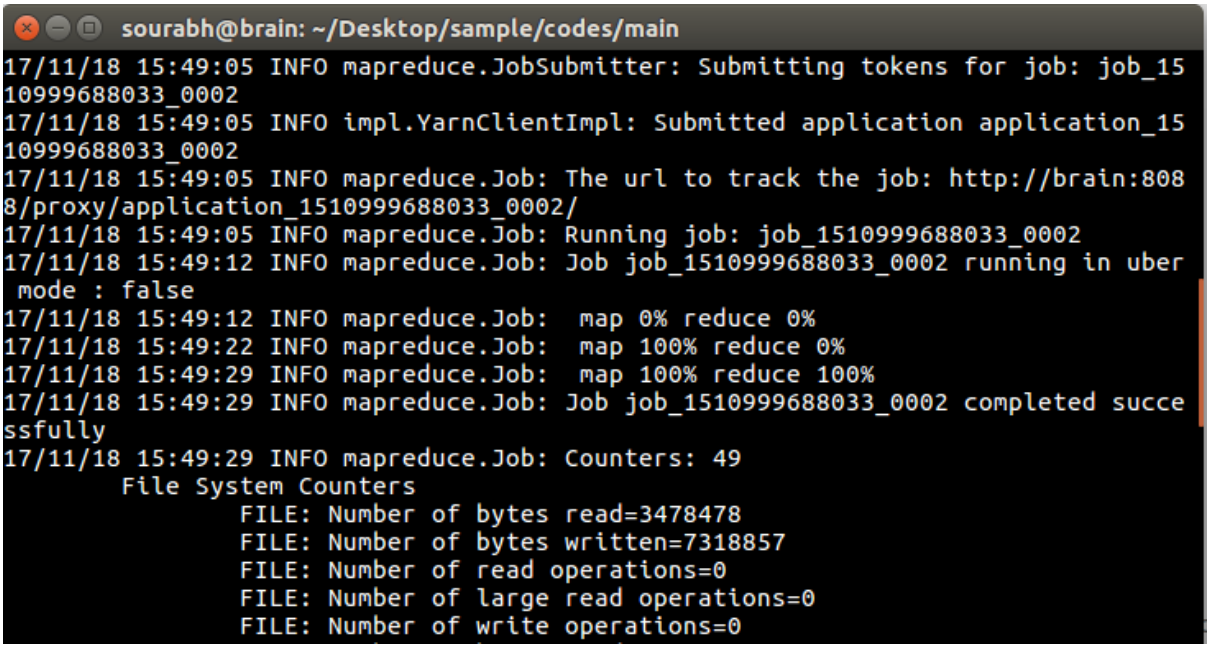


Figure 9.7 streaming job 2 in process

OUTPUT:

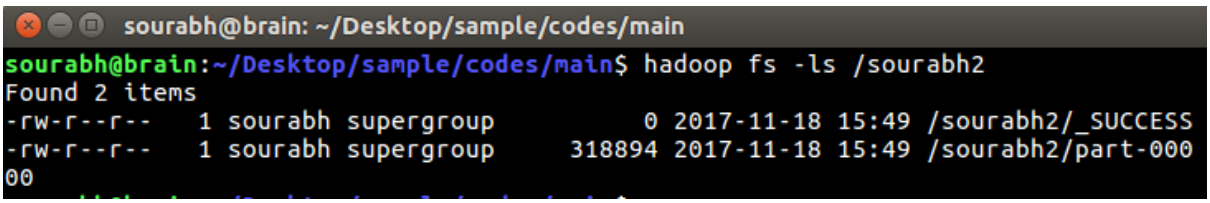


Figure 9.8 verifying output of streaming job2

To verify the job is successful we can check the output directory.

After the complete execution of the streaming command the output can be viewed by the following command:

- $ hadoop fs -cat /sourabh2/part-00000

OR

- Hadoop local server i.e. localhost://50070
- To move the result on the local file system we can pipe the output to a output file.

$ hadoop fs -cat /toptags/part-00000 > student_times.

So on the local file system a file named ‘student_times’ would be created which contains the output of the map-reduce task.

SNAP of the output file:

author_id	Most_active_hours
100000000	9
100000002	4
100000003	5
100000005	1
100000007	3
100000008	16
100000009	20
100000011	3
100000014	4
100000016	23
100000017	5
100000017	22
100000018	2
100000018	3
100000018	11
100000018	16
100000018	19
100000019	5
100000020	5
100000022	17
100000023	3
100000025	3
100000028	4
100000030	14
100000031	3
100000032	1
100000032	4
100000032	19
100000036	4
100000039	4
100000044	5
100000045	4
100000046	5
100000048	4
100000049	14
100000050	4
100000051	23
100000052	10
100000052	12
100000053	0
100000053	6
100000053	16
100000053	17
100000054	3
100000054	18
100000054	20
100000055	4
100000055	5
100000055	6
100000055	17
100000055	18
100000055	22
100000056	12
100000056	16
100000056	20
100000057	8
100000058	4
100000058	17

and so on..

Figure 9.9 output of streaming job2

The output file contains 25,246 entries and it cannot be viewed on this document so just few rows are attached.

9.5 JOB analysis of streaming job 2

Python script:

```
import pandas as pd
import numpy as np

data = pd.read_table('student_times')
author_ids = np.array(data['author_id'])
hours = np.array(data['Most_active_hours'])

hours_count = dict()
for i in hours:
    hours_count[i] = hours_count.get(i,0) + 1
x = sorted(list(hours_count.keys()))
t = np.array(x)
```

```
y = []
for i in x:
    y.append(hours_count[i])

s = pd.Series(times, index = t)

%pylab inline
activ_plt = s.plot(kind = 'bar',title = 'Most Active Hours')
activ_plt.set_xlabel('TIME --->')
activ_plt.set_ylabel('HIT COUNT')
```

9.6 ANALYSIS OUTPUT:

```
Populating the interactive namespace from numpy and matplotlib
<matplotlib.text.Text at 0x7f6f9547b390>
```

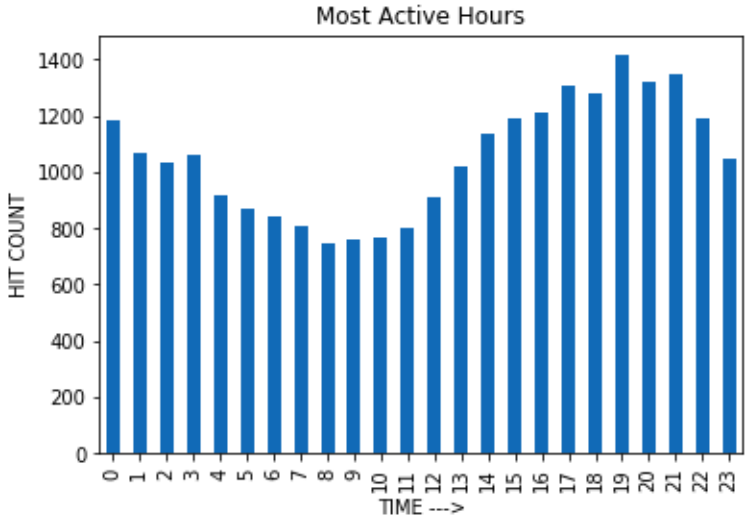


Figure 9.10 Analysis report

CHAPTER 10 HIVE AND PIG

An Overview

- Why we need Pig and Hive?
- Apache Pig.
- PIG Installation
- Pig Basics and some jobs on data-set
- Apache Pig Use Cases -Companies Using Apache Pig.
- Apache Hive.
- How Hive Works.
- Features.
- Hive installation.
- Hive Job
- Pig vs Hive

10.1 – Why we need Pig and Hive?

Google's CEO, Eric Schmidt said: "There were 5 exabytes of information created by the entire world between the dawn of civilization and 2003. Now that same amount is created every two days."

The most impressive thing about this is that mankind is capable of storing, processing and analysing this incredible bulk of data using open source frameworks like Hadoop in reasonable time.

Once data is loaded into Hadoop, the developers have to run alaysis task on it.

There are various ways by which we can do it.

Just solving a word count problem needs at least a 100 lines code, a data mining problem (on wikipedia dataset) would require two MapReduce jobs to be implemented and run sequentially one after the other.

MapReduce in JAVA is very low level. Meaning one needs to deal with all the intricacies and complexities pertaining to JAVA. But not only JAVA programmer, many other programmers like Business Analyst and Data Scientists also want to analyze the data.

Pig and Hive open source alternatives to Hadoop MapReduce were built so that hadoop developers could do the same thing in Java in a less verbose way by writing only fewer lines of code that is easy to understand. Pig, Hive and MapReduce coding approaches are complementary components on the Hadoop stack. A hadoop developer who know Pig and/or Hive does not have to learn Java, however there are certain jobs which can be executed more effectively using Hadoop MapReduce coding approach rather than using Pig or Hive scripts and vice versa.

Requirement:

1. Need of higher level of abstraction on top of MapReduce.

2. Not to deal with low level stuff involved in MapReduce.

This served as the basic motivation for Hive and Pig and numerous other Hadoop projects.

10.2 – Apache Pig.

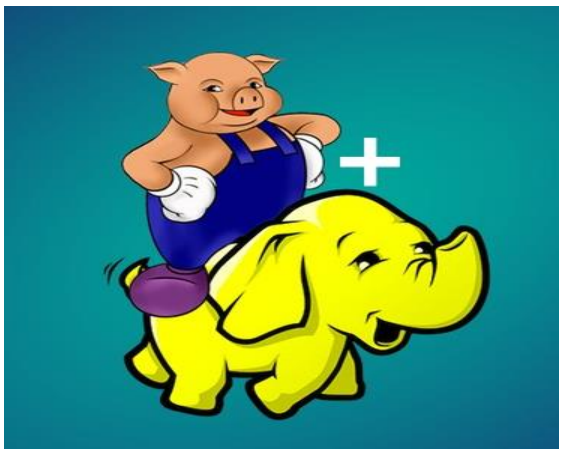


Figure 10.1 – Apache Pig Logo

As Donald Miner, NYC Pig User Group Member rightly said-"If you can do it with Pig, save yourself from the pain because developer time is always worth more than the machine time.

Pig's programming language referred to as Pig Latin is a coding approach that provides high degree of abstraction for MapReduce programming but is a procedural code not declarative. Pig Latin code can be extended through various user defined functions that are written in Python, Java, Groovy, JavaScript, and Ruby. Pig has tools for data storage, data execution and data manipulation. Pig Latin is highly promoted by Yahoo as all the data engineers at Yahoo use Pig for processing data on the biggest hadoop clusters in the world.

Apache Pig is designed to handle any kind of data. Apache Pig is a high level extensible language designed to reduce the complexities of coding MapReduce applications. Pig was developed at Yahoo to help people use Hadoop to emphasize on analysing large unstructured data sets by minimizing the time spent on writing Mapper and Reducer functions.

All tasks are encoded in a manner that helps the system to optimize the execution automatically because **typically 10 lines of code in Pig equal 200 lines of code in Java**. Pig converts its operators into MapReduce code. Apache Pig is composed of 2

components mainly-on is the Pig Latin programming language and the other is the Pig Runtime environment in which Pig Latin programs are executed.

Lets consider a word count problem in java and its equivalent in pig:

.java file:

```
package org.myorg;

import java.io.IOException;

import java.util.*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.*;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

public static class Map extends Mapper {

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

while (tokenizer.hasMoreTokens()) {

word.set(tokenizer.nextToken());

context.write(word, one);

}

}

}

public static class Reduce extends Reducer {
```

```
public void reduce(Text key, Iterable values, Context context)
throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
context.write(key, new IntWritable(sum));
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
Job job = new Job(conf, "wordcount");
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.waitForCompletion(true);
}
}
```

.pig file:

```
input_lines = LOAD '/tmp/word.txt' AS (line:chararray);
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
filtered_words = FILTER words BY word MATCHES '\\w+';
word_groups = GROUP filtered_words BY word;
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS
count, group AS word;
```

```
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/results.txt';

//scripts credit: wiki.
```

The Java MapReduce word count program has 48 lines of java code and also in the very beginning of the code there are ten in-built libraries that have been imported.

If we compare it with its equivalent Pig Latin script shown below then has only 7 lines of code thus making it faster for hadoop developers to code in Pig Latin rather than using Hadoop MapReduce programming approach.

But why name it “Pig”?[10.6]

- Pigs Eat Anything - Pig can operate on data whether it has metadata or not. It can operate on data that is relational, nested, or unstructured. And it can easily be extended to operate on data beyond files, including key/value stores, databases, etc.
- Pigs Live Anywhere - Pig is intended to be a language for parallel data processing. It is not tied to one particular parallel framework. It has been implemented first on Hadoop, but we do not intend that to be only on Hadoop.
- Pigs Are Domestic Animals - Pig is designed to be easily controlled and modified by its users.Pig allows integration of user code where ever possible, so it currently supports user defined field transformation functions, user defined aggregates, and user defined conditionals. These functions can be written in Java or scripting languages that can compile down to Java (e.g. Jython). Pig supports user provided load and store functions. It supports external executables via its stream command and Map Reduce jars via its mapreduce command. It allows users to provide a custom partitioner for their jobs in some circumstances and to set the level of reduce parallelism for their jobs. command. It allows users to set the level of reduce parallelism for their jobs and in some circumstances to provide a custom partitioner.Pig has an optimizer that rearranges some operations in Pig Latin scripts to give better performance, combines Map Reduce jobs together, etc. However, users can easily turn this optimizer off to prevent it from making changes that do not make sense in their situation.

- Pigs Fly - Pig processes data quickly. We want to consistently improve performance, and not implement features in ways that weigh pig down so it can't fly.

Pig vs SQL

In comparison to SQL, Pig

uses lazy evaluation,

uses extract, transform, load (ETL),

is able to store data at any point during a pipeline,

declares execution plans,

supports pipeline splits, thus allowing workflows to proceed along DAGs instead of strictly sequential pipelines.

10.3 - PIG installation.

Step 1: Download Apache Pig tar from:

http://www-eu.apache.org/dist/pig/pig-0.17.0/pig-0.17.0.tar.gz

Apache Pig 0.17.0 is the latest stable version.

Step 2: Extract tar file:

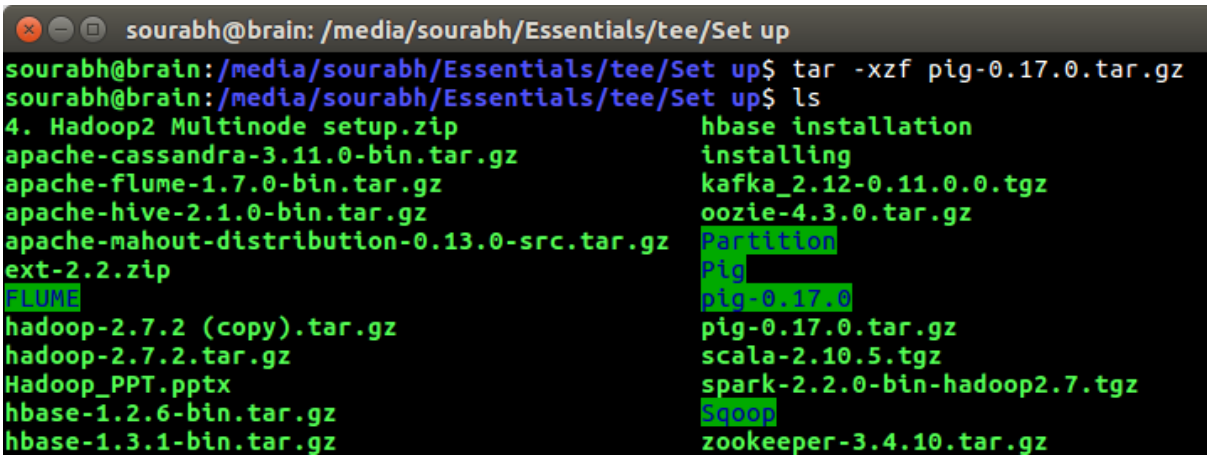


Figure 10.2 - Extracting Pig using tar

Step 3: Make a directory for Pig home and move the files there:

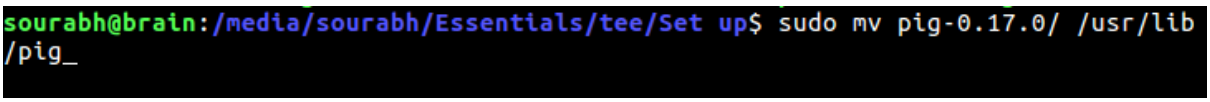


Figure 10.3 - Creating installation directory and moving pig files

Step 4: Edit .bashrc file to save Pig home location:

```
export PIG_HOME=/usr/local/pig-0.17.0
export PATH=$PATH:$PIG_HOME/bin
```

Figure 10.4 - Editing .bashrc file

Step 5: Check for installed Pig version:

```
sourabh@brain:/media/sourabh/Essentials/tee/Set up$ pig -version
Apache Pig version 0.17.0 (r1797386)
compiled Jun 02 2017, 15:41:58
```

Fig 10.5 - Checking Pig version

Step 6: Pig is almost ready, test run in local mode:

```
sourabh@brain: ~/Documents/Hadoop/PIG

grunt> a = load 'NYSE_daily.txt' using TextLoader() as (word:chararray);
2017-11-19 22:53:37,770 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
grunt> describe a;
a: {word: chararray}
grunt> _
```

Figure 10.6 - Pig in local mode

10.4 - Pig Basics.

Pig can be operated in two modes:

local mode - When Local Mode is set up, we can enter using $ pig –x local

map reduce – it can be used by using $ pig command. It run on the cluster.

There are various operators used in pig. Some of them are:

load – it is used to load file.

dump – it is used to display data while using grunt.

Describe – it describe the variable like its data type etc.

foreach & generate – it take data in every pipe.

arithmetic operators – consist of various built in operators like sum(),max() etc.

user defined functions – we can define our own built in udf in pig keeping up with its extensibility philosophy.

Group – it is used to accumulate data into a bag.

Count – count number of distinct data points.

Store – store or save the output into a output file.

10.5 - STREAMING PIG JOB 1

The data set used in this job is a log of purchases across different stores in America. The data contain 6 columns namely:

Date – date of transaction.

Time – time of transaction.

Store – store location from where it was purchased.

Product – type of product purchased.

Cost – cost of item.

Payment – mode of payment.

The data is for complete year i.e. for the entire year 2001!

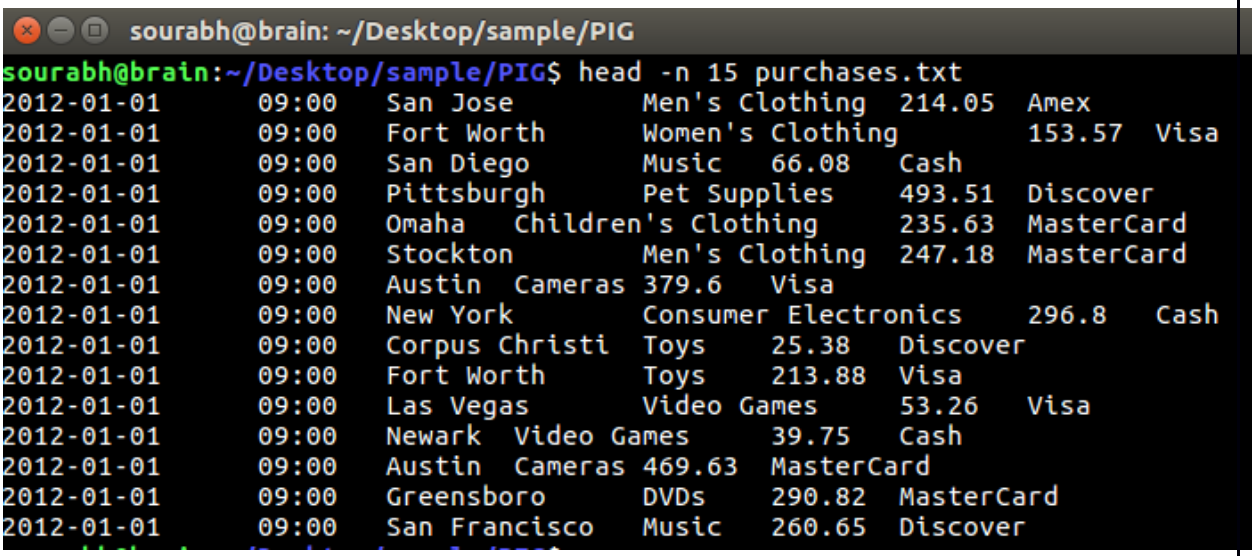


Figure 10.7 – description of dataset

The problem for this job is to analyze the data and find what was the mean sale on Sundays?

PIG scripts: - weekendsale.pig

```
--This script calculate the sale done on specific day of the week for
the complete year
```

```
-- Uses purchases.txt as the input data

-- this script same can be used to find the most loved item brought
or which item are brought at a par1ticular day of the week max. most
popular item


data = load 'purchases.txt' as (date, time, stores, product, cost,
payment);

define mapday `weekendmap.py` ship('weekendmap.py');

-- ship the .py file to filter the data and attach daynumber to it.

fltr_data = stream data through mapday as (date:int, cost:double);

fltr = filter fltr_data by (chararray)date matches '6';

sundaysale = group fltr by date;

-- here the fltr already contains the data only for sunday, group is
optional but we want to calculate mean so if we have a bag we can
easily provide it for calculation

mn = foreach sundaysale generate SUM(fltr.cost)/COUNT(fltr);

-- calculates mean in just one line not to mention the backend
process which split the file etc ....

store mn into
'/home/sourabh/Desktop/sample/PIG/mean_sale_on_sundayF';
```

The script has used weekendmap.py. This script act as a User Defined Function in Pig and this takes data in every pipe and stream it through the UDF. The function of this UDF is to distinguish Sundays and filter the data.

PYTHON script: -

```
#!/usr/bin/env python

from datetime import datetime

import sys


def mapper():

  for line in sys.stdin:

    data = line.strip().split('\t')

    if len(data) == 6 :
```

```
        date, time, store,product, cost, payment = data
        weekday = datetime.strptime(date, "%Y-%m-%d").weekday()
        #print ('{0}\t{1}').format(weekday, cost)
        print(weekday,cost,sep = '\t')

if __name__ == '__main__':
  mapper()
```

TERMINAL COMMAND FOR RUNNING PIGLATIN SCRIPT:

```
$ pig -x local 'weekendsale.pig'
```

JOB RUNNING:

The script is executed with the aid of the above command. The script then splits the data and run job on it based on map-reduce paradigms.

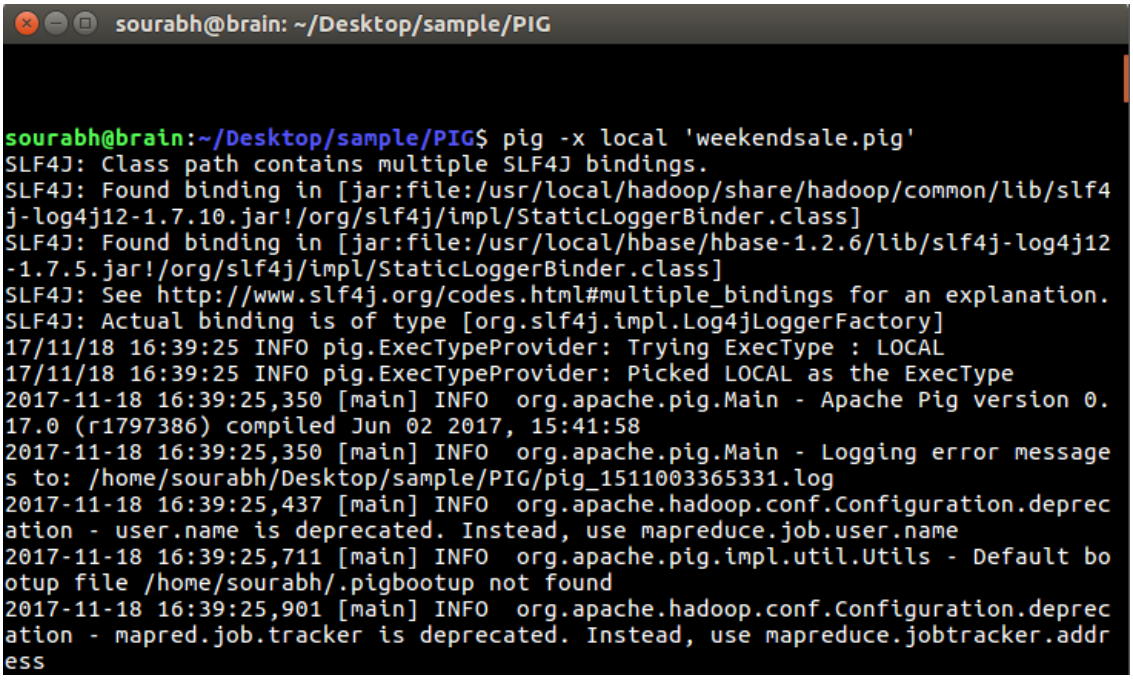


Figure 10.8 – Pig job 1 started

OUTPUT:

The script took 1 minute and 45 seconds to execute as shown in the figure below:

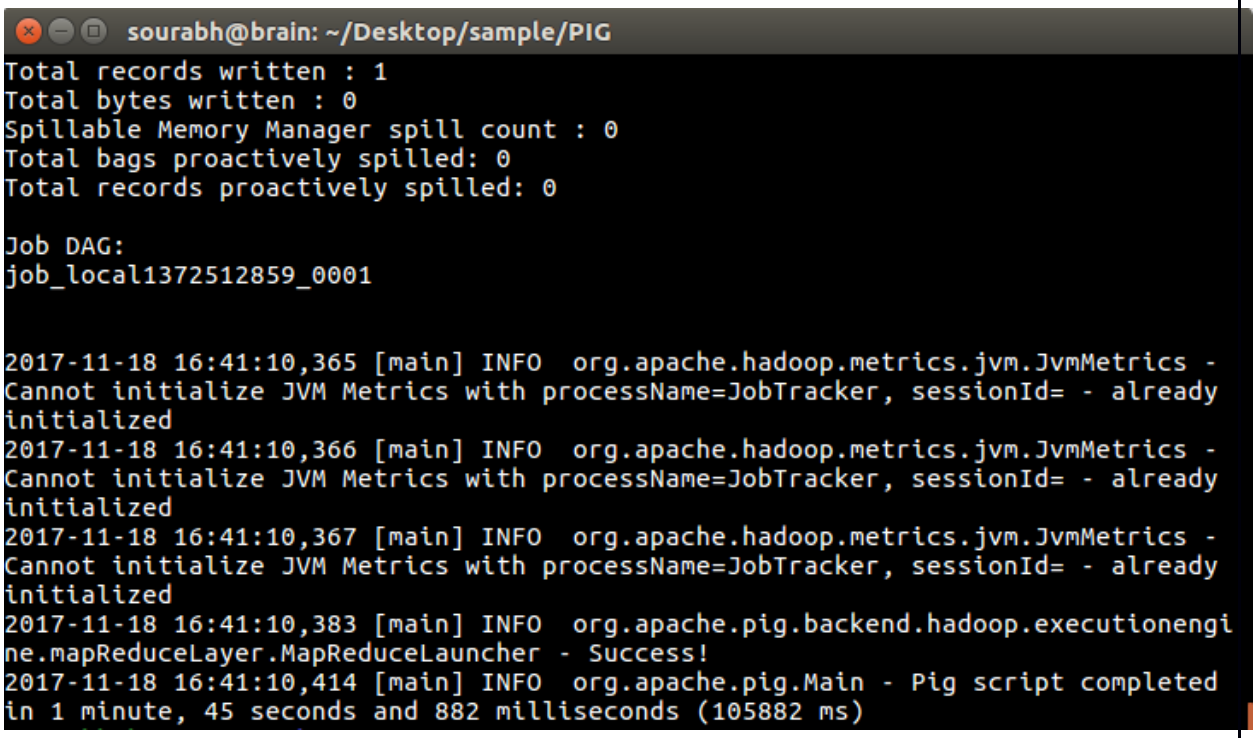


Figure 10.9 – Job 1 done.

The mean sale found on Sunday for the entire year was calculated to be:

249.94644325113643

10.6 - STREAMING JOB 2

The data used in this job is same as used in the python streaming. The discussion board data file named ‘forum_node.tsv’ was used for this script.

The problem for this job was to find the most active hour of each user on the forum. The task is same as previous but this time it has to be done using piglatin.

Piglatin makes the code small and execution smooth and fast.(-tez).

PIG Script: - prep1.pig

```
/*Script to calculate the maximum active time of an user*/

define filterd `auth_id_time.py` ship('auth_id_time.py');

register 'udf.py' using jython as udf;

data = load 'forum_node.tsv';

fltrdata = stream data through filterd as (id, hours:int);

grp_data = group fltrdata by id;

result = foreach grp_data generate group as id,
flatten(udf.gettime(fltrdata,COUNT(fltrdata)));

store result into '/home/sourabh/Desktop/sample/PIG/prep1F';
```

The script uses a User Defined Function written in python named as ‘udf.py’. It also has a streaming function written in python named as ‘auth_id_time.py’.

PYTHON Script: (i) auth_id_time.py (ii) udf.py

(i)

```
#!/usr/bin/env python


import sys,csv


def mapper():
  reader = csv.reader(sys.stdin, delimiter='\t')
  for line in reader:
    author_id = line[3]
    date_added = line[8]
    time = date_added[11:13]
```

```
    #print '{0}\t{1}'.format(author_id, time)
    print(author_id, time, sep = '\t')
if __name__ == '__main__':
  mapper();
```

(ii)

```
@outputSchema('time:{(hr:chararray)}')#It can be confusing when
specifying this structure because it is only need to define what one
element in the bag would look like.
def gettime(data,item_no):
  hours = []
  hrs = {}
  time = []
  for i in range(item_no):
    h = data[i][1]
    hours.append(h)
  for h in hours:
    hrs[h] = hrs.get(h,0) + 1
  active = sorted(hrs.values(), reverse = True)[0]
  for k,v in hrs.items() :
    if v == active :
      time.append(k)
  return time
```

TERMINAL COMMAND FOR RUNNING PIGLATIN SCRIPT:

```
$ pig -x local 'prep1.pig'
```

JOB RUNNING:

The script was executed by the terminal command. It is then executes. The figure below shows execution.

```
sourabh@brain:~/Desktop/sample/PIG$ pig -x local 'prep1.pig'
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/lib/slf4
j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hbase/hbase-1.2.6/lib/slf4j-log4j12
-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
17/11/18 19:10:35 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
17/11/18 19:10:35 INFO pig.ExecTypeProvider: Picked LOCAL as the ExecType
2017-11-18 19:10:35,356 [main] INFO  org.apache.pig.Main - Apache Pig version 0.
17.0 (r1797386) compiled Jun 02 2017, 15:41:58
2017-11-18 19:10:35,356 [main] INFO  org.apache.pig.Main - Logging error message
s to: /home/sourabh/Desktop/sample/PIG/pig_1511012435354.log
2017-11-18 19:10:35,377 [main] INFO  org.apache.hadoop.conf.Configuration.deprec
ation - user.name is deprecated. Instead, use mapreduce.job.user.name
```

Figure 10.10 – Pig job-2 started

OUTPUT:

The script took approximately 26 seconds to complete its execution. The output was saved on the local file system under the folder named ‘prep1F’.

The output file contained 25,245 records. The snap of the output file is shown below:

```
author_id       Most_active_hours
100000000       9
100000002       4
100000003       5
100000005       1
```

Figure10.11 Pig job-2 output

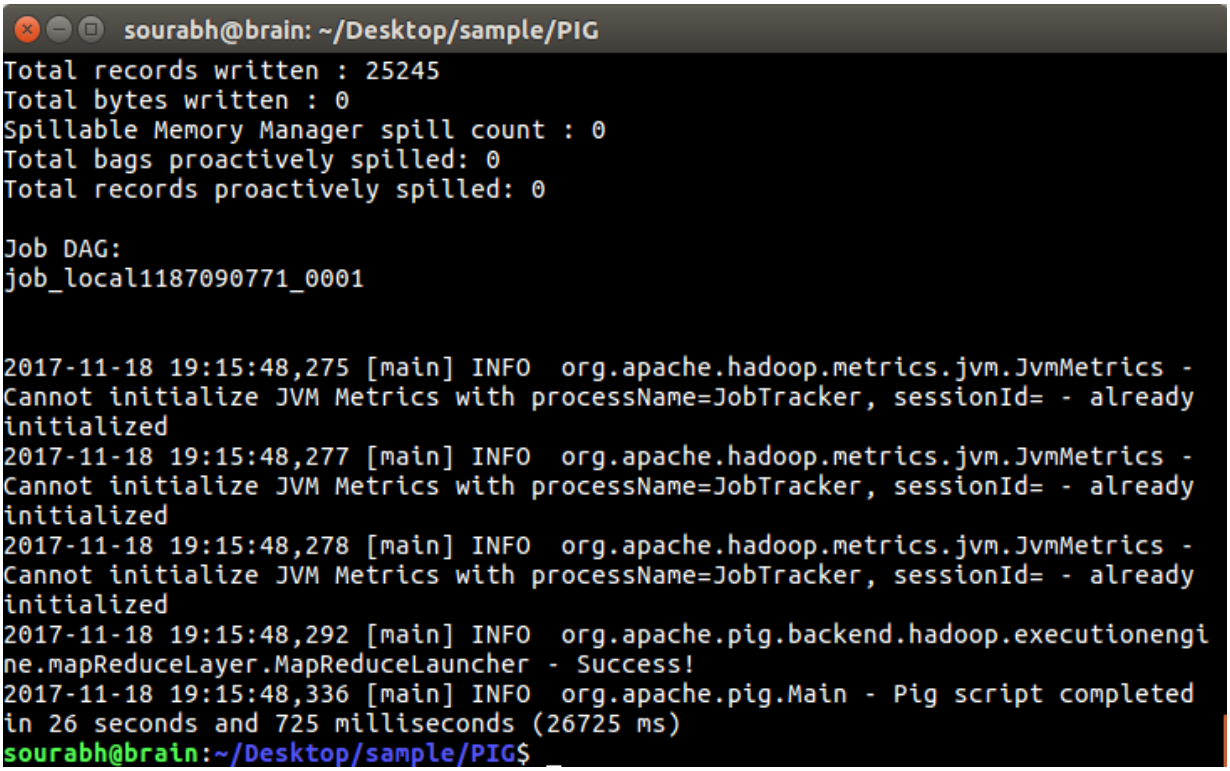


Figure 10.12 – Pig job 2 done

10.7 Apache Pig Use Cases -Companies Using Apache Pig

- Dataium uses Apache Pig to sort and prepare data before it is handed over to MapReduce jobs.
- Apache Pig is an integral part of the "People You May Know" data product at LinkedIn.
- PayPal is a major contributor to the Pig -Eclipse project and uses Apache Pig to analyze transactional data and prevent fraud.

10.8 Apache Hive

Figure 10.13 - Apache Hive Logo

Hadoop was built to organize and store massive amounts of data of all shapes, sizes and formats. Because of Hadoop's "schema on read" architecture, a Hadoop cluster is a perfect reservoir of heterogeneous data—structured and unstructured—from a multitude of sources.

Data analysts use Hive to query, summarize, explore and analyze that data, then turn it into actionable business insight

Hive was created to make it possible for analysts with strong SQL skills (but meager Java programming skills) to run queries on the huge volumes of data that Facebook stored in HDFS.

While initially developed by Facebook, Apache Hive is used and developed by other companies such as Netflix and the Financial Industry Regulatory Authority (FINRA). Amazon maintains a software fork of Apache Hive included in Amazon Elastic MapReduce on Amazon Web Services.

10.9 HOW HIVE WORKS

Hive on LLAP (Live Long and Process) makes use of persistent query servers with intelligent in-memory caching to avoid Hadoop's batch-oriented latency and provide as fast as sub-second query response times against smaller data volumes, while Hive on Tez continues to provide excellent batch query performance against petabyte-scale data sets.

The tables in Hive are similar to tables in a relational database, and data units are organized in a taxonomy from larger to more granular units. Databases are comprised

of tables, which are made up of partitions. Data can be accessed via a simple query language and Hive supports overwriting or appending data.

Within a particular database, data in the tables is serialized and each table has a corresponding Hadoop Distributed File System (HDFS) directory. Each table can be sub-divided into partitions that determine how data is distributed within sub-directories of the table directory. Data within partitions can be further broken down into buckets.

Hive supports all the common primitive data formats such as BIGINT, BINARY, BOOLEAN, CHAR, DECIMAL, DOUBLE, FLOAT, INT, SMALLINT, STRING, TIMESTAMP, and TINYINT. In addition, analysts can combine primitive data types to form complex data types, such as structs, maps and arrays.

10.9.1 Features

Apache Hive supports analysis of large datasets stored in Hadoop's HDFS and compatible file systems such as Amazon S3 filesystem. It provides an SQL-like query language called HiveQL with schema on read and transparently converts queries to MapReduce, Apache Tez and Spark jobs.

Indexing to provide acceleration, index type including compaction and bitmap index as of 0.10, more index types are planned.

Different storage types such as plain text, RCFile, HBase, ORC, and others.

Metadata storage in a relational database management system, significantly reducing the time to perform semantic checks during query execution.

Operating on compressed data stored into the Hadoop ecosystem using algorithms including DEFLATE, BWT, snappy, etc.

Built-in user-defined functions (UDFs) to manipulate dates, strings, and other data-mining tools. Hive supports extending the UDF set to handle use-cases not supported by built-in functions.

SQL-like queries (HiveQL), which are implicitly converted into MapReduce or Tez, or Spark jobs.

10.10 Hive Installation

Step 1: Download Hive tar from:

http://apache.forsale.plus/hive/stable-2/apache-hive-2.3.2-bin.tar.gz

Apache Hive 2.3.2 is the latest stable version.

Step 2: Extract the tar file:

```
sourabh@brain: /media/sourabh/Essentials/tee/Set up
sourabh@brain:/media/sourabh/Essentials/tee/Set up$ tar -xvf apache-hive-2.1.0-b
in.tar.gz
apache-hive-2.1.0-bin/conf/hive-log4j2.properties.template
apache-hive-2.1.0-bin/conf/hive-exec-log4j2.properties.template
apache-hive-2.1.0-bin/conf/beeline-log4j2.properties.template
apache-hive-2.1.0-bin/conf/llap-daemon-log4j2.properties.template
apache-hive-2.1.0-bin/conf/llap-cli-log4j2.properties.template
apache-hive-2.1.0-bin/conf/parquet-logging.properties
```

Figure 10.14 - Extracting Hive using tar command

Step 3: Make directory for Hive home and move the files there:

```
sourabh@brain: /media/sourabh/Essentials/tee/Set up
sourabh@brain:/media/sourabh/Essentials/tee/Set up$ sudo mkdir /usr/lib/hive_
```

Figure 10.15 - Creating installation directory and moving hive files

Step 4: Edit bashrc file to save Hive home location:

```
tee/Set up
/Set up$ gedit ~/.bashrc
```

Figure 10.16 - Editing and saving .bashrc using gedit

```
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
```

Figure 10.17 - .bashrc file with hive_home location

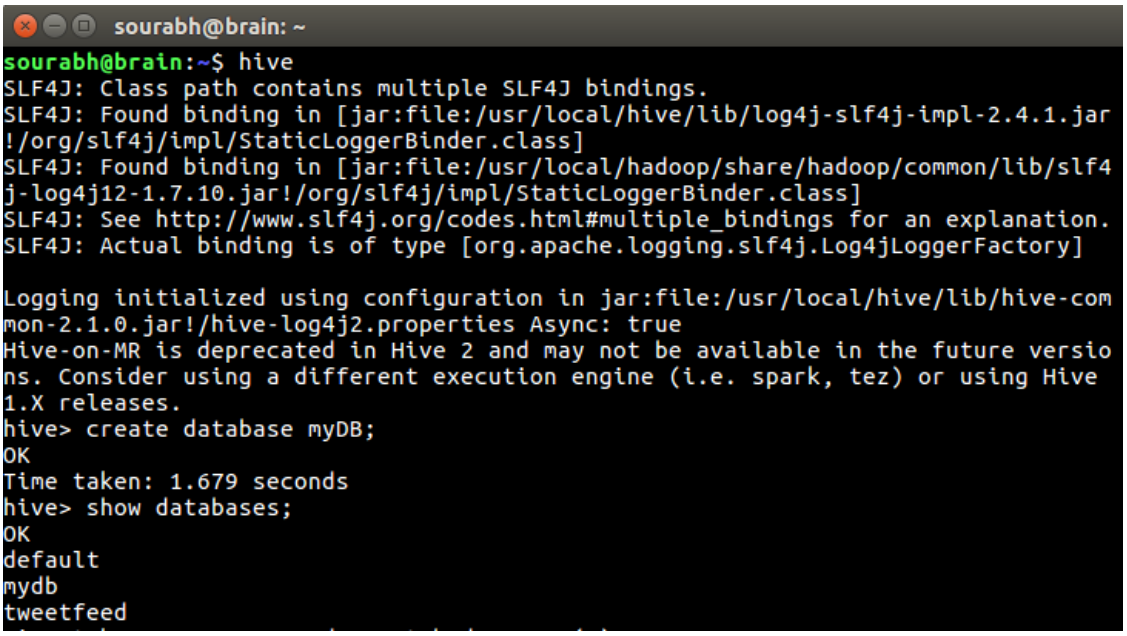
Step 5: Check for the installed Hive version:

```
sourabh@brain: /media/sourabh/Essentials/tee/Set up
sourabh@brain:/media/sourabh/Essentials/tee/Set up$ hive --version
Hive 2.1.0
Subversion git://jcamachguezrMBP/Users/jcamachorodriguez/src/workspaces/hive/HIV
E-release2/hive -r 9265bc24d75ac945bde9ce1a0999fddd8f2aae29
Compiled by jcamachorodriguez on Fri Jun 17 01:03:25 BST 2016
From source with checksum 1f896b8fae57fbd29b047d6d67b75f3c
```

Figure 10.18 - Checking hive version

Step 6: Hive by default uses derby database. Therefore it needs to be initialized first.

Figure 10.19 Running Hive



Step 7: Hive is almost ready, test run:

10.11 Hive Jobs

Job description: Read file 'Crime.csv' from HDFS in HIVE and filter out the Incident ID, Crime Type and District where the crime was reported.

Dataset:

Figure 10.20 – Dataset for Hive

STEPS:

Initialize Hive:

```
$ hive
```

Create a database to use:

```
hive> CREATE DATABASE DB;
hive> USE DB;
```

Create table for the raw data:

```
hive> CREATE TABLE CRIMES
      (DATA STRING);
hive> LOAD DATA INPATH '/Hadoop/Crime.csv'
      INTO TABLE CRIMES;
hive> SELECT * FROM CRIMES;
```

Create table to filter out data:

```
hive> CREATE TABLE CRIME_DATA
      (INCIDENTID INT, CRIME_CLASS INT,
POLICE_DISTRICT STRING);
hive> DESC CRIME_DATA;
```

Filtering data using Regular expression:

```
hive> INSERT OVERWRITE TABLE CRIME_DATA
    SELECT REGEXP_EXTRACT (DATA,'^(?:([^,]*)\,?){1}',1)
INCIDENTID, REGEXP_EXTRACT (DATA,'^(?:([^,]*)\,?){2}',1)
CRIME_CLASS, REGEXP_EXTRACT
(DATA,'^(?:([^,]*)\,?){4}',1) POLICE_DISTRIC, FROM CRIMES;
```

Check the data:

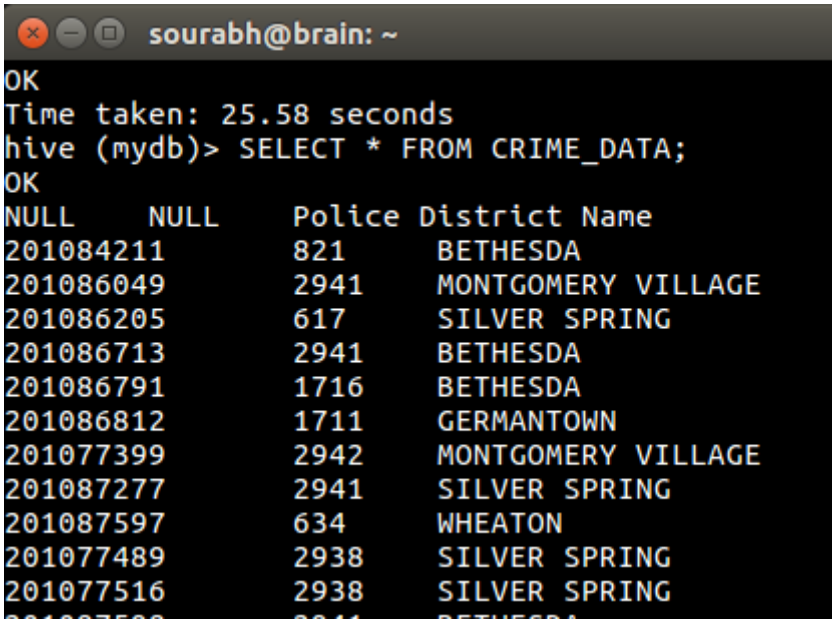


Figure 10.21 Data uploaded in the table

hive> SELECT * FROM CRIME_DATA;

10.12 PIG VS HIVE

Characteristic	Pig	Hive
Language Name	Pig Latin	HiveQL
Type of Language	Dataflow	Declarative (SQL Dialect)
Developed By	Yahoo	Facebook
Data Structures Supported	Nested and Complex	
Relational Complete	YES	YES
Schema Optional	YES	NO

FIGURE 10.22 – PIG V/S HIVE

- Apache Pig is 36% faster than Apache Hive for join operations on datasets.
- Apache Pig is 46% faster than Apache Hive for arithmetic operations.
- Apache Pig is 10% faster than Apache Hive for filtering 10% of the data.
- Apache Pig is 18% faster than Apache Hive for filtering 90% of the data.

CHAPTER 11 HADOOP ECOSYSTEM

An Overview

- Data Integration Components of Hadoop Ecosystem—Sqoop and Flume
- Data Storage Component of Hadoop Ecosystem—HBase
- Monitoring, Management and Orchestration Components of Hadoop Ecosystem- Oozie and Zookeeper
- Apache Kafka

11.1 Data Integration Components of Hadoop Ecosystem- Sqoop and Flume

11.1.1 Sqoop

Sqoop component is used for importing data from external sources into related Hadoop components like HDFS, HBase or Hive. It can also be used for exporting data from Hadoop o other external structured data stores. Sqoop parallelized data transfer, mitigates excessive loads, allows data imports, efficient data analysis and copies data quickly.

Sqoop Use Case

Online Marketer Coupons.com uses Sqoop component of the Hadoop ecosystem to enable transmission of data between Hadoop and the IBM Netezza data warehouse and pipes backs the results into Hadoop using Sqoop.

11.1.2 Flume

Flume component is used to gather and aggregate large amounts of data. Apache Flume is used for collecting data from its origin and sending it back to the resting location (HDFS).Flume accomplishes this by outlining data flows that consist of 3 primary structures channels, sources and sinks. The processes that run the dataflow with flume are known as agents and the bits of data that flow via flume are known as events.

Flume Use Case

Twitter source connects through the streaming API and continuously downloads the tweets (called as events). These tweets are converted into JSON format and sent to the downstream Flume sinks for further analysis of tweets and retweets to engage users on Twitter

11.2 Data Storage Component of Hadoop Ecosystem –HBase

HBase

HBase is a column-oriented database that uses HDFS for underlying storage of data. HBase supports random reads and also batch computations using MapReduce. With HBase NoSQL database enterprise can create large tables with millions of rows and columns on hardware machine. The best practice to use HBase is when there is a requirement for random ‘read or write’ access to big datasets.

HBase Use Case

Facebook is one the largest users of HBase with its messaging platform built on top of HBase in 2010.HBase is also used by Facebook for streaming data analysis, internal monitoring system, Nearby Friends Feature, Search Indexing and scraping data for their internal data warehouses.

11.3 Monitoring, Management and Orchestration Components of Hadoop Ecosystem- Oozie and Zookeeper

11.3.1 Oozie

Oozie is a workflow scheduler where the workflows are expressed as Directed Acyclic Graphs. Oozie runs in a Java servlet container Tomcat and makes use of a database to store all the running workflow instances, their states ad variables along with the workflow definitions to manage Hadoop jobs (MapReduce, Sqoop, Pig and Hive).The workflows in Oozie are executed based on data and time dependencies.

Oozie Use Case

The American video game publisher Riot Games uses Hadoop and the open source tool Oozie to understand the player experience.

11.3.2 Zookeeper-

Zookeeper is the king of coordination and provides simple, fast, reliable and ordered operational services for a Hadoop cluster. Zookeeper is responsible for synchronization service, distributed configuration service and for providing a naming registry for distributed systems.

Zookeeper Use Case-

Found by Elastic uses Zookeeper comprehensively for resource allocation, leader election, high priority notifications and discovery. The entire service of Found built up of various systems that read and write to Zookeeper.

11.4 Apache Kafka

A distributed public-subscribe message developed by LinkedIn that is fast, durable and scalable. Just like other Public-Subscribe messaging systems ,feeds of messages are maintained in topics.

Apache Kafka Use Cases

- Spotify uses Kafka as a part of their log collection pipeline.
- Airbnb uses Kafka in its event pipeline and exception tracking.
- At FourSquare ,Kafka powers online-online and online-offline messaging.
- Kafka power's MailChimp's data pipeline.

Several other common Hadoop ecosystem components include: Avro, Cassandra, Chukwa, Mahout, HCatalog, Ambari and Hama. By implementing Hadoop using one or more of the Hadoop ecosystem components, users can personalize their big data experience to meet the changing business requirements.

The demand for big data analytics will make the elephant stay in the big data room for quite some time.

CONCLUSION

Ninety percent of the data in the world today has been created in the last two years alone. Our current output of data is roughly 2.5 quintillion bytes a day. As the world steadily becomes more connected with an ever-increasing number of electronic devices, this number is only set to grow over the coming years.
Data is useless without the skill to analyze it. With so much of technological advancement in past few years we now have the required software and hardware to process and analyze such humungous volume of data uncover unknown data correlations, hidden consumption patterns, market trends, and consumer preferences, the list is endless and is limited only by human imagination. Hadoop is like a chest of analysis tools for effective data analysis and it continues to expand its array of end-use industries.. There are more job opportunities in Big Data management and Analytics than there before.

Information is the oil in the 21st century and analytic is the combustion engine.

BIBLOGRAPHY

1."Data, data everywhere". The Economist. 25 February 2010. Retrieved 9 December 2012.

2.Hellerstein, Joe (9 November 2008). "Parallel Programming in the Age of Big Data". Gigaom Blog.

3."Welcome to Apache Hadoop!". hadoop.apache.org. Retrieved 2016-08-25.

4.Thomas H. Davenport in Big Data in Big Companies

5.Ng, Andrew Y.; Bradski, Gary; Chu, Cheng-Tao; Olukotun, Kunle; Kim, Sang .Kyun; Lin, Yi-An; Yu, YuanYuan. "Map-Reduce for Machine Learning on Multicore". NIPS 2006

7."How Google Works". baselinemag.com. "As of October, Google was running about 3,000 computing jobs per day through MapReduce, representing thousands of machine-days, according to a presentation by Dean. Among other things, these batch routines analyze the latest Web pages and update Google's indexes."

8.Programming Hive [Book].

9."Pig user defined functions". Retrieved May 3, 2013.

10.As an alternative to Streaming, Python programmers should consider Dumbo, which makes the StreamingMapReduce interface more Pythonic and easier to use

11.IBM Big Data Analytics.

12.Big medical data | MIT News.

13.Hadoop: The Definitive Guide[BOOK]

14.Programming PIG[BOOK]

15.Mapreduce Design Patterns - Donald Miner Adam Shook[BOOK]

.