

Project Report
(Project Semester January-June 2017)

Big Data with Hadoop

Submitted by
Jitender Singh Virk
Student ID: 1302041054

Under the Guidance of

Vijay Prakash Sharma
Assistant Professor-I, CSE
JECRC University

Chaitanya Sharma
Big data Trainer
IBMCE Headstart

Department of Computer Science and Engineering

JECRC University, Jaipur

February to July, 2017

Declaration

I hereby declare that the project work entitled Big Data with Hadoop is an authentic record of my own work carried out at IBMCE Headstart as requirements of six months project semester for the award of degree of B. Tech. Computer Science and Engineering, JECRC University, under the guidance of Chaitanya Sharma and Vijay Prakash Sharma, during February to July, 2017.

Jitender Singh Virk

1302041054

Date:

Certified that the above statement made by the student is correct to the best of our knowledge

Vijay Prakash Sharma
Assistant Professor-I, CSE
JECRC University

Chaitanya Sharma
Big Data Trainer
IBMCE Headstart

Preface

In many ways, this is how I feel about big data and Hadoop. Its inner workings are complex, resting as they do on a mixture of distributed systems theory, practical engineering, and common sense. And to the uninitiated, Hadoop can appear alien.

But it doesn't need to be like this. Stripped to its core, the tools that Hadoop provides for working with big data are simple. If there's a common theme, it is about raising the level of abstraction—to create building blocks for programmers who have lots of data to store and analyze, and who don't have the time, the skill, or the inclination to become distributed systems experts to build the infrastructure to handle it.

With such a simple and generally applicable feature set, it seemed obvious to me when I started using it that Hadoop deserved to be widely used. However, at the time (in early 2006), setting up, configuring, and writing programs to use Hadoop was an art. Things have certainly improved since then: there is more documentation, there are more examples, and there are thriving mailing lists to go to when you have questions. And yet the biggest hurdle for newcomers is understanding what this technology is capable of, where it excels, and how to use it.

The Apache Hadoop community has come a long way. “Big data” has become a household term. In this time, the software has made great leaps in adoption, performance, reliability, scalability, and manageability. The number of things being built and run on the Hadoop platform has grown enormously. In fact, it's difficult for one person to keep track. To gain even wider adoption, I believe we need to make Hadoop even easier to use. This will involve writing more tools; integrating with even more systems; and writing new, improved APIs.

Acknowledgement

The internship opportunity I had with IBMCE Headstar Technologies was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to meet so many wonderful people and professionals who led me through this internship period. I express my deepest thanks to Shumaila Anees, Operations Head for taking part in useful decision & giving necessary advices and guidance and arranged all facilities to make life easier. I choose this moment to acknowledge her contribution gratefully.

It is my radiant sentiment to place on record my best regards, deepest sense of gratitude to Mr. Chaitanya Sharma (Training guide) for their careful and precious guidance which were extremely valuable for my study both theoretically and practically.

I perceive as this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, to attain desired career objectives. Hope to continue cooperation with all of you in the future.

Sincerely,

Jitender Singh Virk

1302041054

Date:

Table of Contents

Abstract

| | |
|--|-----------|
| 1. Introduction..... | 1 |
| 1.1 Data..... | 1 |
| 1.2 Data Storage and Analysis..... | 2 |
| 1.3 A Brief History of Apache Hadoop..... | 3 |
| 2. Hadoop MapReduce | 6 |
| 2.1 A Weather Dataset..... | 6 |
| 2.2 Analysing the data with unix tools..... | 8 |
| 2.3 Analysing the data with Hadoop..... | 10 |
| 2.4 Hadoop Streaming | 11 |
| 3. Hadoop Distributed File System(HDFS) | 14 |
| 3.1 What is Distributed File System..... | 14 |
| 3.2 Why Is a Block in HDFS So Large..... | 15 |
| 3.3 Rack awareness | 17 |
| 3.4 Writes on HDFS..... | 18 |
| 3.5 Reads on HDFS..... | 19 |
| 4. Apache Flume..... | 22 |
| 4.1 What is Flume..... | 22 |
| 4.2 Apache Flume Architecture | 23 |
| 4.3 Additional Components of Flume Agent..... | 25 |
| 4.4 Configuration | 26 |
| 5. Apache Pig | 31 |
| 5.1 What is Apache Pig | 31 |
| 5.2 Apache Architecture | 33 |
| 5.3 Grunt Shell..... | 35 |
| 5.4 Pig Latin..... | 38 |
| 5.5 Load and Store Operators | 39 |
| 5.6 Grouping and Joining..... | 43 |
| 5.7 Filtering..... | 45 |
| 5.8 Sorting..... | 46 |

| | |
|-------------------------|-----------|
| 6. Apache Hive | 47 |
| 6.1 What is Hive | 47 |
| 6.2 Working of Hive | 48 |
| 6.3 Data Types | 50 |
| 6.4 Hive Statements | 51 |
| 6.5 Build in Operators | 56 |
| 6.6 HiveQL | 56 |
| | |
| 7. Final Project | 59 |
| 7.1 Problem Statement | 60 |
| 7.2 Methodology | 62 |
| | |
| 8. Conclusion | 70 |
| 9. Learnings | 71 |
| 10. Appendices | 72 |
| 11. References | 74 |

Abstract

In today's highly developed world, every minute, people around the globe express themselves via various platforms on the Web. And in each minute, a huge amount of unstructured data is generated. This data is in the form of text which is gathered from forums and social media websites. Such data is termed as big data. User opinions are related to a wide range of topics like politics, latest gadgets and products. These opinions can be mined using various technologies and are of utmost importance to make predictions or for one-to-one consumer marketing since they directly convey the viewpoint of the masses. Here we propose to analyse the sentiments of Twitter users through their tweets in order to extract what they think. Hence we are using Hadoop for sentiment analysis which will process the huge amount of data on a Hadoop cluster faster.

Sentiment analysis is extremely useful in social media monitoring as it allows us to gain an overview of the wider public opinion behind certain topics. Social media monitoring tools like Brand-watch Analytics make that process quicker and easier than ever before. The applications of sentiment analysis are broad and powerful. The ability to extract insights from social data is a practice that is being widely adopted by organizations across the world. Shifts in sentiment on social media have been shown to correlate with shifts in the stock market. The Obama administration used sentiment analysis to gauge public opinion to policy announcements and campaign messages ahead of 2012 presidential election.

The ability to quickly understand consumer attitudes and react accordingly is something that Expedia Canada took advantage of when they noticed that there was a steady increase in negative feedback to the music used in one of their television adverts.

Chapter 1

INTRODUCTION

1.1 Data!

We live in the data age. It's not easy to measure the total volume of data stored electronically, but an IDC estimate put the size of the "digital universe" at 4.4 zettabytes in 2013 and is forecasting a tenfold growth by 2020 to 44 zettabytes.¹ A zettabyte is 10^{21} bytes, or equivalently one thousand exabytes, one million petabytes, or one billion terabytes. That's more than one disk drive for every person in the world.

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.

So there's a lot of data out there. But you are probably wondering how it affects you. Most of the data is locked up in the largest web properties (like search engines) or in scientific or financial institutions, isn't it? Does the advent of big data affect smaller organizations or individuals?

More generally, the digital streams that individuals are producing are growing apace. Microsoft Research's MyLifeBits project gives a glimpse of the archiving of personal information that may become commonplace in the near future. MyLifeBits was an experiment where an individual's interactions—phone calls, emails, documents—were captured electronically and stored for later access. The data gathered included a photo taken every minute, which resulted in an overall data volume of 1 gigabyte per month.

When storage costs come down enough to make it feasible to store continuous audio and video, the data volume for a future MyLifeBits service will be many times that. The trend is for every individual's data footprint to grow, but perhaps more significantly, the amount of data generated by machines as a part of the Internet of Things will be even greater than that generated by people. Machine logs, RFID readers, sensor networks, vehicle GPS traces, retail

transactions—all of these contribute to the growing mountain of data. The volume of data being made publicly available increases every year, too. Organizations no longer have to merely manage their own data; success in the future will be dictated to a large extent by their ability to extract value from other organizations' data. Initiatives such as Public Data Sets on Amazon Web Services and Infochimps.org exist to foster the “information commons,” where data can be freely (or for a modest price) shared for anyone to download and analyze. Mashups between different information sources make for unexpected and hitherto unimaginable applications.

Take, for example, the Astrometry.net project, which watches the Astrometry group on Flickr for new photos of the night sky. It analyzes each image and identifies which part of the sky it is from, as well as any interesting celestial bodies, such as stars or galaxies. This project shows the kinds of things that are possible when data (in this case, tagged photographic images) is made available and used for something (image analysis) that was not anticipated by the creator. It has been said that “more data usually beats better algorithms,” which is to say that for some problems (such as recommending movies or music based on past preferences), however fiendish your algorithms, often they can be beaten simply by having more data (and a less sophisticated algorithm). The good news is that big data is here. The bad news is that we are struggling to store and analyse it.

1.2 Data Storage and Analysis

The problem is simple: although the storage capacities of hard drives have increased massively over the years, access speeds—the rate at which data can be read from drives— have not kept up. One typical drive from 1990 could store 1,370 MB of data and had a transfer speed of 4.4 MB/s, so you could read all the data from a full drive in around five minutes. Over 20 years later, 1-terabyte drives are the norm, but the transfer speed is around 100 MB/s, so it takes more than two and a half hours to read all the data off the disk.

This is a long time to read all data on a single drive—and writing is even slower. The obvious way to reduce the time is to read from multiple disks at once. Imagine if we had 100 drives, each holding one hundredth of the data. Working in parallel, we could read the data in under two minutes.

Using only one hundredth of a disk may seem wasteful. But we can store 100 datasets, each of which is 1 terabyte, and provide shared access to them. We can imagine that the users of such a system would be happy to share access in return for shorter analysis times, and statistically, that their analysis jobs would be likely to be spread over time, so they wouldn't interfere with

each other too much. There's more to being able to read and write data in parallel to or from multiple disks, though.

The first problem to solve is hardware failure: as soon as you start using many pieces of hardware, the chance that one will fail is fairly high. A common way of avoiding data loss is through replication: redundant copies of the data are kept by the system so that in the event of failure, there is another copy available. This is how RAID works, for instance, although Hadoop's filesystem, the Hadoop Distributed Filesystem (HDFS), takes a slightly different approach, as you shall see later.

1.3 A Brief History of Apache Hadoop

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project.

The Origin of the Name "Hadoop":

The name Hadoop is not an acronym; it's a made-up name. The project's creator, Doug Cutting, explains how the name came about:

The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term. Projects in the Hadoop ecosystem also tend to have names that are unrelated to their function, often with an elephant or other animal theme ("Pig," for example). Smaller components are given more descriptive (and therefore more mundane) names. This is a good principle, as it means you can generally work out what something does from its name. For example, the namenode8 manages the filesystem namespace. Building a web search engine from scratch was an ambitious goal, for not only is the software required to crawl and index websites complex to write, but it is also a challenge to run without a dedicated operations team, since there are so many moving parts. It's expensive, too: Mike Cafarella and Doug Cutting estimated a system supporting a one-billion-page index would cost around \$500,000 in hardware, with a monthly running cost of \$30,000.⁹ Nevertheless, they believed it was a worthy goal, as it would open up and ultimately democratize search engine algorithms. Nutch was started in 2002, and a working crawler and search system quickly emerged.

However, its creators realized that their architecture wouldn't scale to the billions of pages on the Web. Help was at hand with the publication of a paper in 2003 that described the architecture of Google's distributed filesystem, called GFS, which was being used in production at Google.¹⁰ GFS, or something like it, would solve their storage needs for the very

large files generated as a part of the web crawl and indexing process. In particular, GFS would free up time being spent on administrative tasks such as managing storage nodes. In 2004, Nutch's developers set about writing an open source implementation, the Nutch Distributed Filesystem (NDFS).

In 2004, Google published the paper that introduced MapReduce to the world. Early in 2005, the Nutch developers had a working MapReduce implementation in Nutch, and by the middle of that year all the major Nutch algorithms had been ported to run using MapReduce and NDFS. NDFS and the MapReduce implementation in Nutch were applicable beyond the realm of search, and in February 2006 they moved out of Nutch to form an independent subproject of Lucene called Hadoop. At around the same time, Doug Cutting joined Yahoo!, which provided a dedicated team and the resources to turn Hadoop into a system that ran at web scale (see the following sidebar). This was demonstrated in February 2008 when Yahoo! announced that its production search index was being generated by a 10,000-core Hadoop cluster. In January 2008, Hadoop was made its own top-level project at Apache, confirming its success and its diverse, active community. By this time, Hadoop was being used by many other companies besides Yahoo!, such as Last.fm, Facebook, and the *New York Times*. In one well-publicized feat, the *New York Times* used Amazon's EC2 compute cloud to crunch through 4 terabytes of scanned archives from the paper, converting them to PDFs for the Web. The processing took less than 24 hours to run using 100 machines, and the project probably wouldn't have been embarked upon without the combination of Amazon's pay-by-the-hour model (which allowed the *NYT* to access a large number of machines for a short period) and Hadoop's easy-to-use parallel programming model.

In April 2008, Hadoop broke a world record to become the fastest system to sort an entire terabyte of data. Running on a 910-node cluster, Hadoop sorted 1 terabyte in 209 seconds (just under 3.5 minutes), beating the previous year's winner of 297 seconds. In November of the same year, Google reported that its MapReduce implementation sorted 1 terabyte in 68 seconds.¹⁵ Then, in April 2009, it was announced that a team at Yahoo! had used Hadoop to sort 1 terabyte in 62 seconds. The trend since then has been to sort even larger volumes of data at ever faster rates. In the 2014 competition, a team from Databricks were joint winners of the Gray Sort benchmark. They used a 207-node Spark cluster to sort 100 terabytes of data in 1,406 seconds, a rate of 4.27 terabytes per minute. Today, Hadoop is widely used in mainstream enterprises. Hadoop's role as a generalpurpose storage and analysis platform for big data has been recognized by the industry, and this fact is reflected in the number of products that use or incorporate Hadoop in some way. Commercial Hadoop support is available from large,

established enterprise vendors, including EMC, IBM, Microsoft, and Oracle, as well as from specialist Hadoop companies such as Cloudera, Hortonworks, and MapR.

Chapter 2

HADOOP MAPREDUCE

MapReduce is a programming model for data processing. The model is simple, yet not too simple to express useful programs in. Hadoop can run MapReduce programs written in various languages; Most importantly, MapReduce programs are inherently parallel, thus putting very large-scale data analysis into the hands of anyone with enough machines at their disposal. MapReduce comes into its own for large datasets, so let's start by looking at one.

2.1 A Weather Dataset

For our example, we will write a program that mines weather data. Weather sensors collect data every hour at many locations across the globe and gather a large volume of log data, which is a good candidate for analysis with MapReduce because we want to process all the data, and the data is semi-structured and record-oriented.

Data Format

The data we will use is from the National Climatic Data Center, or NCDC. The data 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 are always present and are of fixed width.

Example 2-1 shows a sample line with some of the salient fields annotated. The line has been split into multiple lines to show each field; in the real file, fields are packed into one line with no delimiters.

0057

332130 # USAF weather station identifier

99999 # WBAN weather station identifier

19500101 # observation date

0300 # observation time

4

+51317 # latitude (degrees x 1000)
 +028783 # longitude (degrees x 1000)
 FM-12
 +0171 # elevation (meters)
 99999
 V020
 320 # wind direction (degrees)
 1 # quality code
 N
 0072
 1
 00450 # sky ceiling height (meters)
 1 # quality code
 C
 N
 010000 # visibility distance (meters)
 1 # quality code
 N
 9
 -0128 # air temperature (degrees Celsius x 10)
 1 # quality code
 -0139 # dew point temperature (degrees Celsius x 10)
 1 # quality code
 10268 # atmospheric pressure (hectopascals x 10)
 1 # quality code

Datafiles are organized by date and weather station. There is a directory for each year from 1901 to 2001, each containing a gzipped file for each weather station with its readings for that year. For example, here are the first entries for 1990:

```

% ls raw/1990 | head
010010-99999-1990.gz
010014-99999-1990.gz
010015-99999-1990.gz
010016-99999-1990.gz
010017-99999-1990.gz
  
```

010030-99999-1990.gz

010040-99999-1990.gz

010080-99999-1990.gz

010100-99999-1990.gz

010150-99999-1990.gz

There are tens of thousands of weather stations, so the whole dataset is made up of a large number of relatively small files. It's generally easier and more efficient to process a smaller number of relatively large files, so the data was pre-processed so that each year's readings were concatenated into a single file.

2.2 Analyzing the Data with Unix Tools

What's the highest recorded global temperature for each year in the dataset? We will answer this first without using Hadoop, as this information will provide a performance baseline and a useful means to check our results. The classic tool for processing line-oriented data is *awk*.

Example 2-2 is a small script to calculate the maximum temperature for each year.

Example 2-2. A program for finding the maximum recorded temperature by year from NCDC weather records

```
#!/usr/bin/env bash
```

```
for year in all/*
```

```
do
```

```
echo -ne `basename $year .gz`"\t"
```

```
gunzip -c $year | \
```

```
awk '{ temp = substr($0, 88, 5) + 0;
```

```
q = substr($0, 93, 1);
```

```
if (temp !=9999 && q ~ /[01459]/ && temp > max) max = temp }
```

```
END { print max }'
```

```
Done
```

The script loops through the compressed year files, first printing the year, and then processing each file using *awk*. The *awk* script extracts two fields from the data: the air temperature and the quality code. The air temperature value is turned into an integer by adding 0. Next, a test is applied to see whether the temperature is valid (the value 9999 signifies a missing value in the NCDC dataset) and whether the quality code indicates that the reading is not suspect or

erroneous. If the reading is OK, the value is compared with the maximum value seen so far, which is updated if a new maximum is found. The END block is executed after all the lines in the file have been processed, and it prints the maximum value.

Here is the beginning of a run:

```
% ./max_temperature.sh
```

```
1901 317
```

```
1902 244
```

```
1903 289
```

```
1904 256
```

```
1905 283
```

```
...
```

The temperature values in the source file are scaled by a factor of 10, so this works out as a maximum temperature of 31.7°C for 1901 (there were very few readings at the beginning of the century, so this is plausible). The complete run for the century took 42 minutes in one run on a single EC2 High-CPU Extra Large instance. To speed up the processing, we need to run parts of the program in parallel. In theory, this is straightforward: we could process different years in different processes, using all the available hardware threads on a machine. There are a few problems with this, however.

First, dividing the work into equal-size pieces isn't always easy or obvious. In this case, the file size for different years varies widely, so some processes will finish much earlier than others. Even if they pick up further work, the whole run is dominated by the longest file. A better approach, although one that requires more work, is to split the input into fixed-size chunks and assign each chunk to a process.

Second, combining the results from independent processes may require further processing. In this case, the result for each year is independent of other years, and they may be combined by concatenating all the results and sorting by year. If using the fixed-size chunk approach, the combination is more delicate. For this example, data for a particular year will typically be split into several chunks, each processed independently. We'll end up with the maximum temperature for each chunk, so the final step is to look for the highest of these maximums for each year.

Third, you are still limited by the processing capacity of a single machine. If the best time you can achieve is 20 minutes with the number of processors you have, then that's it. You can't make it go faster. Also, some datasets grow beyond the capacity of a single Machine. When we start using multiple machines, a whole host of other factors come into play, mainly falling

into the categories of coordination and reliability. Who runs the overall job? How do we deal with failed processes?

So, although it's feasible to parallelize the processing, in practice it's messy. Using a framework like Hadoop to take care of these issues is a great help.

2.3 Analysing the Data with Hadoop

To take advantage of the parallel processing that Hadoop provides, we need to express our query as a MapReduce job. After some local, small-scale testing, we will be able to run it on a cluster of machines.

Map and Reduce

MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. The programmer also specifies two functions: the map function and the reduce function. The input to our map phase is the raw NCDC data. We choose a text input format that gives us each line in the dataset as a text value. The key is the offset of the beginning of the line from the beginning of the file, but as we have no need for this, we ignore it.

Our map function is simple. We pull out the year and the air temperature, because these are the only fields we are interested in. In this case, the map function is just a data preparation phase, setting up the data in such a way that the reduce function can do its work on it: finding the maximum temperature for each year. The map function is also a good place to drop bad records: here we filter out temperatures that are missing, suspect, or erroneous. To visualize the way the map works, consider the following sample lines of input data (some unused columns have been dropped to fit the page, indicated by ellipses):

0067011990999991950051507004...9999999N9+00001+9999999999...

0043011990999991950051512004...9999999N9+00221+9999999999...

0043011990999991950051518004...9999999N9-00111+9999999999...

0043012650999991949032412004...0500001N9+01111+9999999999...

0043012650999991949032418004...0500001N9+00781+9999999999...

These lines are presented to the map function as the key-value pairs:

(0, 006701199099999**1950**051507004...9999999N9+**00001**+9999999999...)

(106, 004301199099999**1950**051512004...9999999N9+**00221**+9999999999...)

(212, 004301199099999**1950**051518004...9999999N9-**00111**+9999999999...)

(318, 004301265099999**1949**032412004...0500001N9+**01111**+9999999999...)

(424, 004301265099999**1949**032418004...0500001N9+**00781**+9999999999...)

The keys are the line offsets within the file, which we ignore in our map function. The map function merely extracts the year and the air temperature (indicated in bold text), and emits them as its output (the temperature values have been interpreted as integers):

```
(1950, 0)
(1950, 22)
(1950, -11)
(1949, 111)
(1949, 78)
```

The output from the map function is processed by the MapReduce framework before being sent to the reduce function. This processing sorts and groups the key-value pairs by key. So, continuing the example, our reduce function sees the following input:

```
(1949, [111, 78])
(1950, [0, 22, -11])
```

Each year appears with a list of all its air temperature readings. All the reduce function has to do now is iterate through the list and pick up the maximum reading:

```
(1949, 111)
(1950, 22)
```

This is the final output: the maximum global temperature recorded in each year. The whole data flow is illustrated in [Figure 2-1](#). At the bottom of the diagram is a Unix pipeline, which mimics the whole MapReduce flow and which we will see again later in this chapter when we look at Hadoop Streaming.

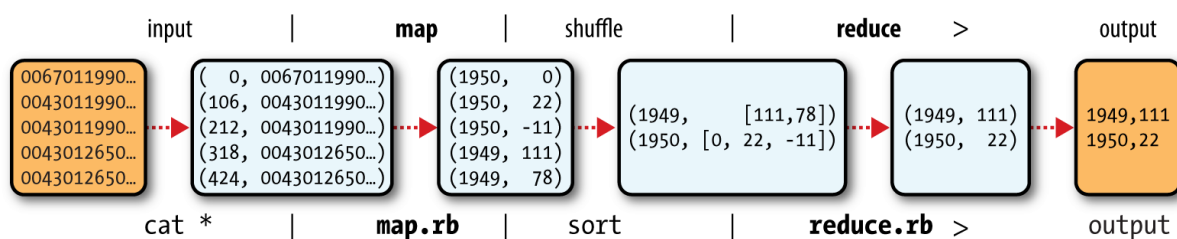


Figure 2-1. MapReduce logical data flow

2.4 Hadoop Streaming

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. 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. Let's illustrate this by rewriting our MapReduce program for finding maximum temperatures by year in Streaming.

Python

Streaming supports any programming language that can read from standard input and write to standard output, so for readers more familiar with Python, here's the same example again.⁵ The map script is in [Example 2-9](#), and the reduce script is in [Example 2-10](#).

Example 2-9. Map function for maximum temperature in Python

```
#!/usr/bin/env python
import re
import sys
for line in sys.stdin:
    val = line.strip()
    (year, temp, q) = (val[15:19], val[87:92], val[92:93])
    if (temp != "+9999" and re.match("[01459]", q)):
        print "%s\t%s" % (year, temp)
```

Example 2-10. Reduce function for maximum temperature in Python

```
#!/usr/bin/env python
import sys
(last_key, max_val) = (None, -sys.maxint)
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)
```

We can test the programs and run the job in the same way we did in Ruby. For example, to run a test:

```
% cat input/ncdc/sample.txt | \  
ch02-mr-intro/src/main/python/max_temperature_map.py | \  
sort | ch02-mr-intro/src/main/python/max_temperature_reduce.py
```

```
1949 111  
1950 22
```

Chapter 3

HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

3.1 What is a Distributed File System?

A DFS is a file system which uses multiple machines (nodes), working in coordination to each other, to store data and allow access to multiple users through multiple nodes. This collection of machines is known as a cluster.

DFS uses network bandwidth for making different nodes to interact with each other. It is like a single system working on multiple computers in which the files and directories are made global and appear identical from any node in the cluster.

For example : If there are 3 nodes in a cluster N1, N2, N3, a user inserts data in the DFS from N1, it would be accessible to N2 and N3 with full authority as if it is present on them.

HDFS runs on top of local file system, it is a logically created over physical storage present on the various machines in the Hadoop cluster.

HDFS is designed to tolerate high component failure rate by replicating the files over multiple machines.

Further, it is designed only to handle large files as the larger the file the less it has to seek the data into various machines.

It Deals with Streaming or sequential data rather than random access, sequential data means few seeks (Hadoop only seeks to the beginning block and read sequentially).

HDFS has two main layers:

- **Namespace** manages directories, files and blocks. It supports file system operations such as creation, modification, deletion and listing of files and directories.

Block Storage has two parts:

- **Block Management** maintains the membership of datanodes in the cluster. It supports block-related operations such as creation, deletion, modification and getting location of the blocks. It also takes care of replica placement and replication.

- **Physical Storage** stores the blocks and provides read/write access to it.

HDFS follows a master-slave architecture and merely has 2 components:

1. **DataNodes (Slave)**

- Datanode act as the slave in hadoop distributed file system.
- Datanodes keep communicating with Namenode by sending a periodic heartbeat signal to it signifying that they are alive.
- Datanodes are the place where actual data reside.
- should have lots of hard disks space because actual data is stored here.

Block

A disk has a block size, which is the minimum amount of data that it can read or write.

Filesystems for a single disk build on this by dealing with data in blocks, which are an integral multiple of the disk block size

HDFS, too, has the concept of a block, but it is a much larger unit—128 MB by default.

Like in a filesystem for a single disk, files in HDFS are broken into block-sized chunks, which are stored as independent units. Unlike a filesystem for a single disk, a file in

HDFS that is smaller than a single block does not occupy a full block's worth of underlying storage. (For example, a 1 MB file stored with a block size of 128 MB uses 1 MB of disk space, not 128 MB.).

3.2 Why Is a Block in HDFS So Large?

HDFS blocks are large compared to disk blocks, and the reason is to minimize the cost of seeks. If the block is large enough, the time it takes to transfer the data from the disk can be significantly longer than the time to seek to the start of the block. Thus, transferring a large file made of multiple blocks operates at the disk transfer rate.

A quick calculation shows that if the seek time is around 10 ms and the transfer rate is 100 MB/s, to make the seek time 1% of the transfer time, we need to make the block size around 100 MB. The default is actually 128 MB, although many HDFS installations use larger block sizes. This figure will continue to be revised upward as transfer speeds grow with new generations of disk drives.

This argument shouldn't be taken too far, however. Map tasks in MapReduce normally operate on one block at a time, so if you have too few tasks

Benefits:

- a file can be larger than any single disk in the network
- Second, making the unit of abstraction a block rather than a file simplifies the storage subsystem. Blocks fit well with replication for providing fault tolerance and availability.
- To insure against corrupted blocks and disk and machine failure, each block is replicated to a small number of physically separate machines (typically three). If a block becomes unavailable, a copy can be read from another location in a way that is transparent to the client.

When you store a file in HDFS, the system breaks it down into a set of individual **blocks** and stores these blocks in various DataNodes in the Hadoop cluster.

Default block size of hadoop is 128 MB. HDFS has no idea (and doesn't care) what's stored inside the file, so raw files are not split in accordance with rules that we humans would understand. So it just splits the file after 128 MB.

For example: a 300 MB file will be saved in following ways:

Block 1 > 128 MB

Block 2 > 128 MB

Block 3 > 44 MB

In the above case the remaining 84MB of the Block 3 is not wasted. The filesystem is not physically divided into blocks (128MB). It's just an abstraction to store the metadata in the NameNode. HDFS is installed on the existing file system, 1 hadoop block consist of numerous OS file blocks, which are much smaller than the Hadoop blocks.

2. **NameNode** (Master)

- Namenode serves as the master in hadoop distributed file system.
- Name node store the metadata of the system. Metadata consist of
 - ◆ Information of files and their replicated data

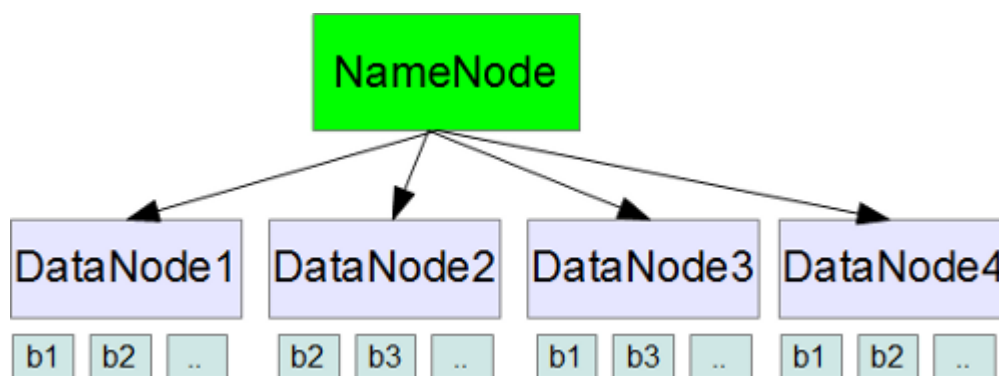
◆ Information about blocks

◆ Information about datanodes

→ Namenode does not store actual data to be processed.

→ Name node should be a highly configured machine.

→ RAM in the Namenode should be High because metadata is highly accessed so it requires high throughput.



How does hadoop provide fault tolerance?

Hadoop framework replicates the data on more than 1 datanode, so that if one datanode is down, the data can be retrieved from another node. By default, Hadoop has a replication factor of 3 that means it replicates the file on 2 more datanodes. This replication factor can be changed.

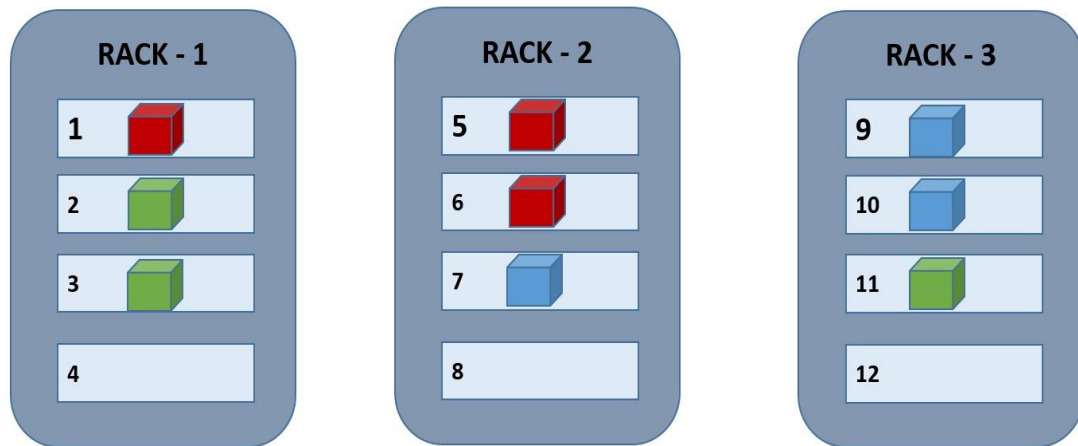
3.3 Rack awareness: Rack awareness is a concept by which Hadoop framework determines the data nodes on which the data is to be replicated.

Rack: A rack is a collection of multiple datanodes that are physically near and are all connected to the same network switch. Nodes on same rack exhibits higher network bandwidth as compared to nodes on different racks. A Hadoop Cluster is a collection of racks.

Block A : 

Block B : 

Block B : 



If the namenode has to save the data on across different racks. This prevents losing the data when entire rack fails.

On a multi-rack cluster, block replications are maintained with a policy that no more than one replica is placed on one node and no more than two replicas are placed on the same rack with a constraint that the number of racks used for block replication should always be less than the total number of block replicas.

When a new block is created, the first replica is placed on the local node, the next replica is made on a different rack, and the third replica is made on a different node on the second rack.

Schema on read:

Hadoop follows schema on read, this means that it saves the file as it is without verifying the data. It verifies the data while performing read operation. This allows faster writes in hadoop.

3.4 Writes on HDFS:

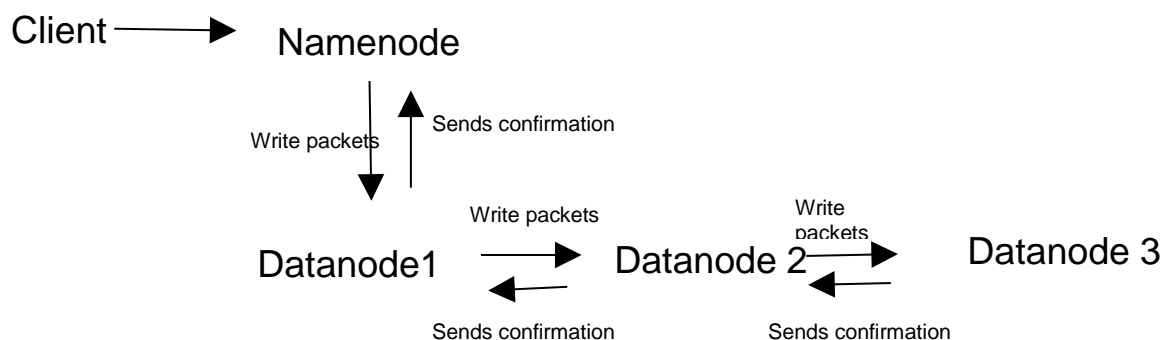
Client server generates a *create* request to the Namenode. Then the Namenode checks if the file does not already exists (because Namenode has the metadata).

- If the file already exist, then hadoop framework throws an error, because data is precious and Hadoop does not allow overwriting of data.

- If the file does not exist, then namenode determine the first block to be written. If there is space present on the DataNode where the client is present, it chooses that datanode otherwise randomly chooses Datanode to save the data and further chooses 2 DataNodes to replicate the data..

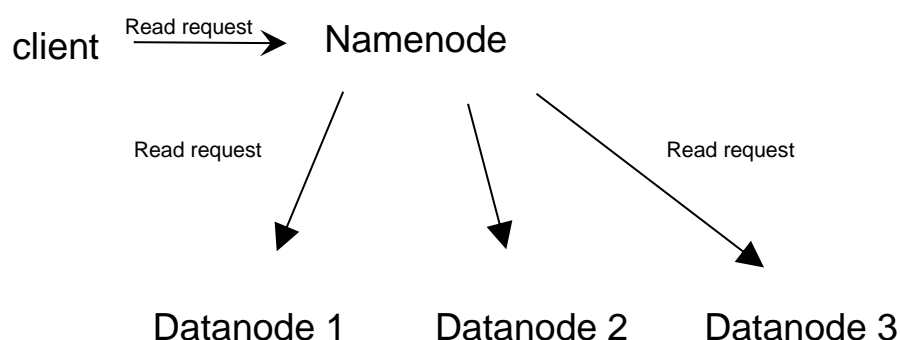
Datanode 1 writes on itself and forward the write request to Datanode 2 chosen by the namenode, which in turn saves the data and forwards the write request to the next DataNode.

After all writes are done the datanode 3 sends a confirmation to datanode 2 that it has saved the data which sends a confirmation to datanode 1, which gives a confirmation to the namenode and then the metadata of the system is updated.



3.5 Reads on HDFS :

While reading the file datanode can directly retrieve the data from any of the datanode available at the moment.



HDFS metadata broadly breaks down into 2 categories of files:

fsimage – An fsimage file contains the complete state of the file system at a point in time.

Every file system modification is assigned a unique, monotonically increasing transaction ID.

An fsimage file represents the file system state after all modifications up to a specific transaction ID.

Edit logs – An edit log is a file that lists each file system change (file creation, deletion or modification) that was made after the most recent fsimage. Only during the restart of namenode, edit logs are applied to fsimage to get the latest snapshot of the file system.

But namenode restart are rare in production clusters.

This result in following problems:

1. Editlog become very large, which will be challenging to manage it.
2. Namenode restart takes long time because lot of changes has to be merged.
3. In the case of crash, we will lost huge amount of metadata since fsimage is very old So to overcome this issues we need a mechanism which will help us reduce the edit log size which is manageable and have up to date fsimage ,so that load on namenode reduces.

Secondary Namenode

Secondary Namenode helps to overcome the above issues by taking over responsibility of merging editlogs with fsimage from the namenode.

1. It gets the edit logs from the namenode in regular intervals and applies to fsimage
 2. Once it has new fsimage, it copies back to namenode
 3. Namenode will use this fsimage for the next restart,which will reduce the startup time
- Secondary Namenode whole purpose is to have a checkpoint in HDFS. It is just a helper node for namenode.That's why it also known as checkpoint node inside the community.

High availability (HA)

Formerly, if a cluster had a single NameNode, and that machine or process became unavailable, the entire cluster would be unavailable until the NameNode was either restarted or started on a separate machine. This situation impacted the total availability of the HDFS cluster in two major ways:

- In the case of an unplanned event such as a machine crash, the cluster would be unavailable until an operator restarted the NameNode.
- Planned maintenance events such as software or hardware upgrades on the NameNode machine would result in periods of cluster downtime.

Hadoop, overall, has always had a robust and failure-tolerant architecture, with the exception of this key area. Therefore, NameNode was referred to as single point of failure (SPOF). To overcome this issue an additional NameNode called Standby NameNode was introduced in Hadoop version 2.

Standby NameNode

Standby NameNode provides hot backup for NameNode. At any point in time, exactly one of the NameNodes is in an Active state, and the other is in a Standby state. The Active NameNode is responsible for all client operations in the cluster, while the Standby is simply acting as a slave, maintaining enough state to provide a fast failover if necessary.

In order for the Standby Namenode to keep its state synchronized with the Active Namenode, both nodes communicate with a group of separate daemons called **Journal Nodes**.

When any namespace modification is performed by the Active node, it durably logs a record of the modification to a majority of these JNs. The Standby node reads these edits from the JNs and applies them to its own name space.

In the event of a failover, the Standby will ensure that it has read all of the edits from the JournalNodes before promoting itself to the Active state. This ensures that the namespace state is fully synchronized before a failover occurs.

It is vital for an HA cluster that only one of the NameNodes is Active at a time.

HDFS Federation

In order to scale the name service horizontally, federation uses multiple independent namenodes/namespaces. The namenodes are federated, that is, the namenodes are independent and don't require coordination with each other. The datanodes are used as common storage for blocks by all the namenodes. Each datanode registers with all the namenodes in the cluster. Datanodes send periodic heartbeats and block reports and handle commands from the namenodes.

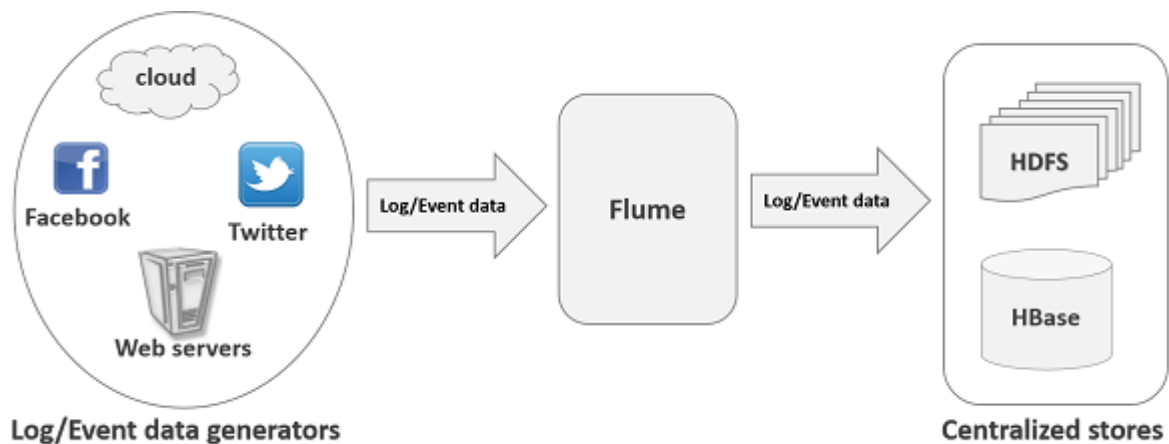
A **Block Pool** is a set of blocks that belong to a single namespace. Datanodes store blocks for all the block pools in the cluster. A Namespace and its block pool together are called **Namespace Volume**.

Chapter 4

APACHE FLUME

4.1 What is Flume?

Apache Flume is a tool/service/data ingestion mechanism for collecting aggregating and transporting large amounts of streaming data such as log files, events (etc...) from various sources to a centralized data store. Flume is a highly reliable, distributed, and configurable tool. It is principally designed to copy streaming data (log data) from various web servers to HDFS.



4.1.1 Applications of Flume

Assume an e-commerce web application wants to analyze the customer behavior from a particular region. To do so, they would need to move the available log data in to Hadoop for analysis. Here, Apache Flume comes to our rescue. Flume is used to move the log data generated by application servers into HDFS at a higher speed.

4.1.2 Advantages of Flume

Here are the advantages of using Flume –

- Using Apache Flume we can store the data in to any of the centralized stores (HBase, HDFS).

- When the rate of incoming data exceeds the rate at which data can be written to the destination, Flume acts as a mediator between data producers and the centralized stores and provides a steady flow of data between them.
- Flume provides the feature of **contextual routing**.
- The transactions in Flume are channel-based where two transactions (one sender and one receiver) are maintained for each message. It guarantees reliable message delivery.
- Flume is reliable, fault tolerant, scalable, manageable, and customizable.

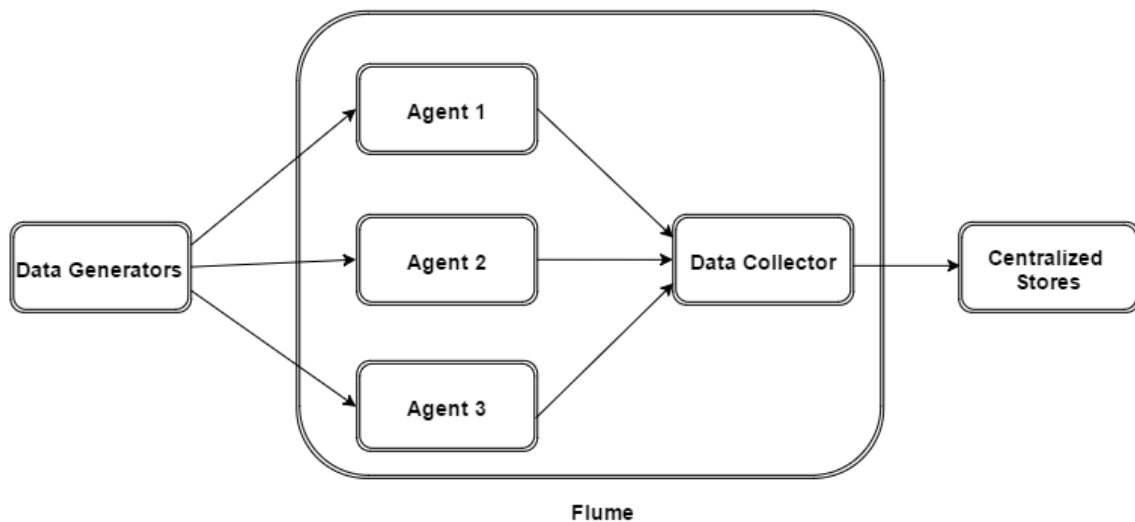
4.1.3 Features of Flume

Some of the notable features of Flume are as follows –

- Flume ingests log data from multiple web servers into a centralized store (HDFS, HBase) efficiently.
- Using Flume, we can get the data from multiple servers immediately into Hadoop.
- Along with the log files, Flume is also used to import huge volumes of event data produced by social networking sites like Facebook and Twitter, and e-commerce websites like Amazon and Flipkart.
- Flume supports a large set of sources and destinations types.
- Flume supports multi-hop flows, fan-in fan-out flows, contextual routing, etc.
- Flume can be scaled horizontally.

4.2 Apache Flume Architecture

The following illustration depicts the basic architecture of Flume. As shown in the illustration, **data generators** (such as Facebook, Twitter) generate data which gets collected by individual Flume **agents** running on them. Thereafter, a **data collector** (which is also an agent) collects the data from the agents which is aggregated and pushed into a centralized store such as HDFS or HBase.



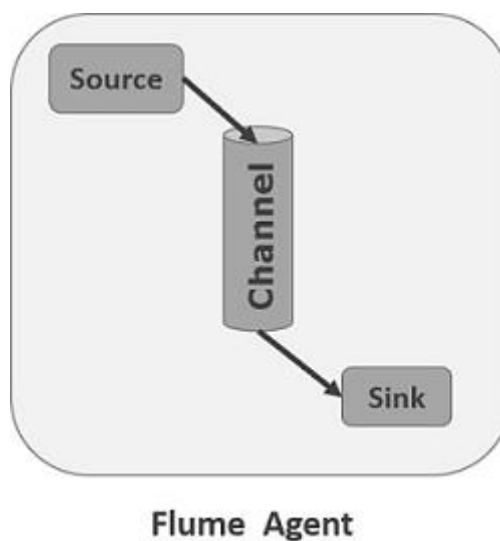
4.2.1 Flume Event

An **event** is the basic unit of the data transported inside **Flume**. It contains a payload of byte array that is to be transported from the source to the destination accompanied by optional headers. A typical Flume event would have the following structure –



4.2.2 Flume Agent

An **agent** is an independent daemon process (JVM) in Flume. It receives the data (events) from clients or other agents and forwards it to its next destination (sink or agent). Flume may have more than one agent. Following diagram represents a **Flume Agent**



As shown in the diagram a Flume Agent contains three main components namely, **source**, **channel**, and **sink**.

4.2.3 Source

A **source** is the component of an Agent which receives data from the data generators and transfers it to one or more channels in the form of Flume events. Apache Flume supports several types of sources and each source receives events from a specified data generator.

Example – Avro source, Thrift source, twitter 1% source etc.

4.2.4 Channel

A **channel** is a transient store which receives the events from the source and buffers them till they are consumed by sinks. It acts as a bridge between the sources and the sinks. These channels are fully transactional and they can work with any number of sources and sinks.

Example – JDBC channel, File system channel, Memory channel, etc.

4.2.5 Sink

A **sink** stores the data into centralized stores like HBase and HDFS. It consumes the data (events) from the channels and delivers it to the destination. The destination of the sink might be another agent or the central stores.

Example – HDFS sink

Note – A flume agent can have multiple sources, sinks and channels. We have listed all the supported sources, sinks, channels in the Flume configuration chapter of this tutorial.

4.3 Additional Components of Flume Agent

What we have discussed above are the primitive components of the agent. In addition to this, we have a few more components that play a vital role in transferring the events from the data generator to the centralized stores.

4.3.1 Interceptors

Interceptors are used to alter/inspect flume events which are transferred between source and channel.

4.3.2 Channel Selectors

These are used to determine which channel is to be opted to transfer the data in case of multiple channels. There are two types of channel selectors –

- **Default channel selectors** – These are also known as replicating channel selectors they replicates all the events in each channel.
- **Multiplexing channel selectors** – These decides the channel to send an event based on the address in the header of that event.

4.3.3 Sink Processors

These are used to invoke a particular sink from the selected group of sinks. These are used to create failover paths for your sinks or load balance events across multiple sinks from a channel.

4.4 Configuration

In the Flume configuration file, we need to –

- Name the components of the current agent.
- Describe/Configure the source.
- Describe/Configure the sink.
- Describe/Configure the channel.
- Bind the source and the sink to the channel.

Usually we can have multiple agents in Flume. We can differentiate each agent by using a unique name. And using this name, we have to configure each agent.

4.4.1 Naming the Components

First of all, you need to name/list the components such as sources, sinks, and the channels of the agent, as shown below.

```
agent_name.sources = source_name  
agent_name.sinks = sink_name  
agent_name.channels = channel_name
```

Flume supports various sources, sinks, and channels. They are listed in the table given below.

| Sources | Channels | Sinks |
|---|---|--|
| <ul style="list-style-type: none"> • Avro Source • Thrift Source • Exec Source • JMS Source • Spooling Directory Source • Twitter 1% firehose Source • Kafka Source • NetCat Source • Sequence Generator Source • Syslog Sources • Syslog TCP Source • Multiport Syslog TCP Source • Syslog UDP Source • HTTP Source • Stress Source • Legacy Sources • Thrift Legacy Source • Custom Source • Scribe Source | <ul style="list-style-type: none"> • Memory Channel • JDBC Channel • Kafka Channel • File Channel • Spillable Memory Channel • Pseudo Transaction Channel | <ul style="list-style-type: none"> • HDFS Sink • Hive Sink • Logger Sink • Avro Sink • Thrift Sink • IRC Sink • File Roll Sink • Null Sink • HBaseSink • AsyncHBaseSink • MorphlineSolrSink • ElasticSearchSink • Kite Dataset Sink • Kafka Sink |

You can use any of them. For example, if you are transferring Twitter data using Twitter source through a memory channel to an HDFS sink, and the agent name id **TwitterAgent**, then

```
TwitterAgent.sources = Twitter
TwitterAgent.channels = MemChannel
TwitterAgent.sinks = HDFS
```

After listing the components of the agent, you have to describe the source(s), sink(s), and channel(s) by providing values to their properties.

4.4.2 Describing the Source

Each source will have a separate list of properties. The property named “type” is common to every source, and it is used to specify the type of the source we are using.

Along with the property “type”, it is needed to provide the values of all the **required** properties of a particular source to configure it, as shown below.

```
agent_name.sources. source_name.type = value
agent_name.sources. source_name.property2 = value
agent_name.sources. source_name.property3 = value
```

For example, if we consider the **twitter source**, following are the properties to which we *must* provide values to configure it.

```
TwitterAgent.sources.Twitter.type = Twitter (type name)
TwitterAgent.sources.Twitter.consumerKey =
TwitterAgent.sources.Twitter.consumerSecret =
TwitterAgent.sources.Twitter.accessToken =
TwitterAgent.sources.Twitter.accessTokenSecret =
```

4.4.3 Describing the Sink

Just like the source, each sink will have a separate list of properties. The property named “type” is common to every sink, and it is used to specify the type of the sink we are using. Along with the property “type”, it is needed to provide values to all the **required** properties of a particular sink to configure it, as shown below.

```
agent_name.sinks. sink_name.type = value
agent_name.sinks. sink_name.property2 = value
```

```
agent_name.sinks.sink_name.property3 = value
```

For example, if we consider **HDFS sink**, following are the properties to which we *must* provide values to configure it.

```
TwitterAgent.sinks.HDFS.type = hdfs (type name)
```

```
TwitterAgent.sinks.HDFS.hdfs.path = HDFS directory's Path to store the data
```

4.4.4 Describing the Channel

Flume provides various channels to transfer data between sources and sinks. Therefore, along with the sources and the channels, it is needed to describe the channel used in the agent.

To describe each channel, you need to set the required properties, as shown below.

```
agent_name.channels.channel_name.type = value
```

```
agent_name.channels.channel_name.property2 = value
```

```
agent_name.channels.channel_name.property3 = value
```

For example, if we consider **memory channel**, following are the properties to which we *must* provide values to configure it.

```
TwitterAgent.channels.MemChannel.type = memory (type name)
```

4.4.5 Binding the Source and the Sink to the Channel

Since the channels connect the sources and sinks, it is required to bind both of them to the channel, as shown below.

```
agent_name.sources.source_name.channels = channel_name
```

```
agent_name.sinks.sink_name.channels = channel_name
```

The following example shows how to bind the sources and the sinks to a channel. Here, we consider **twitter source**, **memory channel**, and **HDFS sink**.

```
TwitterAgent.sources.Twitter.channels = MemChannel
```

```
TwitterAgent.sinks.HDFS.channels = MemChannel
```

4.4.6 Starting a Flume Agent

After configuration, we have to start the Flume agent. It is done as follows –

```
$ bin/flume-ng agent --conf ./conf/ -f conf/twitter.conf  
Dflume.root.logger=DEBUG,console -n TwitterAgent
```

where –

- **agent** – Command to start the Flume agent
- **--conf , -c <conf>** – Use configuration file in the conf directory
- **-f <file>** – Specifies a config file path, if missing
- **--name, -n <name>** – Name of the twitter agent
- **-D property =value** – Sets a Java system property value.

Chapter 5

APACHE PIG

5.1 What is Apache Pig?

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with **Hadoop**; we can perform all the data manipulation operations in Hadoop using Apache Pig. To write data analysis programs, Pig provides a high-level language known as **Pig Latin**. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data. To analyze data using **Apache Pig**, programmers need to write scripts using Pig Latin language. All these scripts are internally converted to Map and Reduce tasks. Apache Pig has a component known as **Pig Engine** that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

Why Do We Need Apache Pig?

Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

- Using **Pig Latin**, programmers can perform MapReduce tasks easily without having to type complex codes in Java.
- Apache Pig uses **multi-query approach**, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.
- Pig Latin is **SQL-like language** and it is easy to learn Apache Pig when you are familiar with SQL.
- Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

Features of Pig

Apache Pig comes with the following features –

- **Rich set of operators** – It provides many operators to perform operations like join, sort, filter, etc.
- **Ease of programming** – Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
- **Optimization opportunities** – The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.
- **Extensibility** – Using the existing operators, users can develop their own functions to read, process, and write data.
- **UDF's** – Pig provides the facility to create **User-defined Functions** in other programming languages such as Java and invoke or embed them in Pig Scripts.
- **Handles all kinds of data** – Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

Applications of Apache Pig

Apache Pig is generally used by data scientists for performing tasks involving ad-hoc processing and quick prototyping. Apache Pig is used –

- To process huge data sources such as web logs.
- To perform data processing for search platforms.
- To process time sensitive data loads.

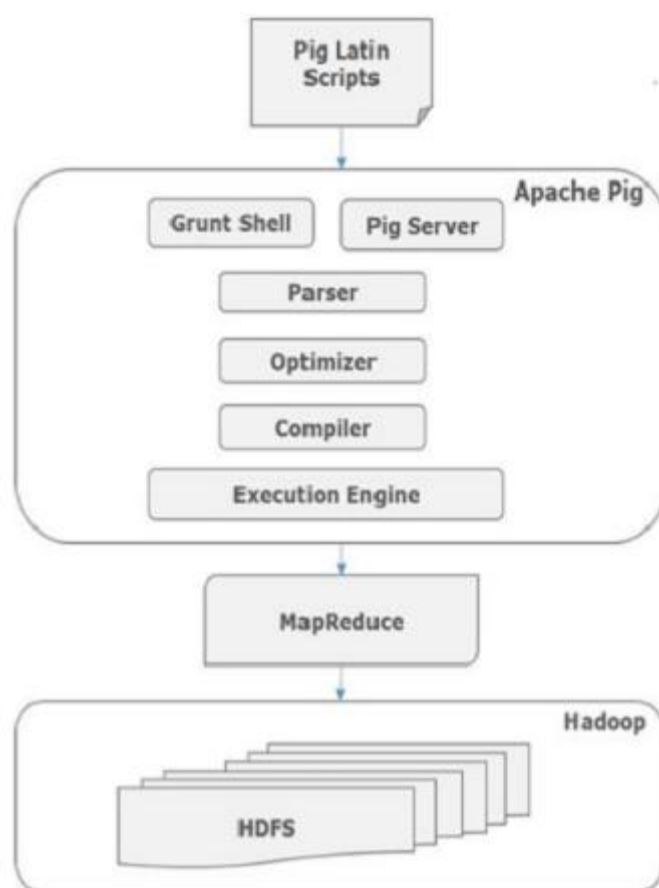
Apache Pig – History

In **2006**, Apache Pig was developed as a research project at Yahoo, especially to create and execute MapReduce jobs on every dataset. In **2007**, Apache Pig was open sourced via Apache incubator. In **2008**, the first release of Apache Pig came out. In **2010**, Apache Pig graduated as an Apache top-level project.

5.2 Apache Architecture

The language used to analyze data in Hadoop using Pig is known as **Pig Latin**. It is a highlevel data processing language which provides a rich set of data types and operators to perform various operations on the data. To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.

Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer's job easy. The architecture of Apache Pig is shown below.



Apache Pig Components

As shown in the figure, there are various components in the Apache Pig framework. Let us take a look at the major components.

Parser

Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.

In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

Optimizer

The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.

Compiler

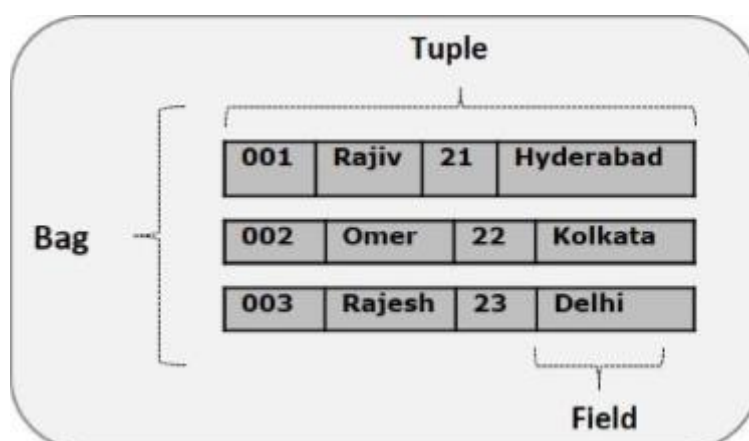
The compiler compiles the optimized logical plan into a series of MapReduce jobs.

Execution engine

Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

Pig Latin Data Model

The data model of Pig Latin is fully nested and it allows complex non-atomic datatypes such as **map** and **tuple**. Given below is the diagrammatical representation of Pig Latin's data model.



Atom

Any single value in Pig Latin, irrespective of their data, type is known as an **Atom**. It is stored as string and can be used as string and number. int, long, float, double, chararray, and bytearray are the atomic values of Pig. A piece of data or a simple atomic value is known as a **field**. **Example** – ‘raja’ or ‘30’

Tuple

A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type. A tuple is similar to a row in a table of RDBMS. **Example** – (Raja, 30)

Bag

A bag is an unordered set of tuples. In other words, a collection of tuples (non-unique) is known as a bag. Each tuple can have any number of fields (flexible schema). A bag is represented by ‘{}’. It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contain the same number of fields or that the fields in the same position (column) have the same type. **Example** – {(Raja, 30), (Mohammad, 45)}

A bag can be a field in a relation; in that context, it is known as **inner bag**.

Example – {Raja, 30, {9848022338, raja@gmail.com,}}

Map

A map (or data map) is a set of key-value pairs. The **key** needs to be of type chararray and should be unique. The **value** might be of any type. It is represented by ‘[]’

Example – [name#Raja, age#30]

Relation

A relation is a bag of tuples. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

5.3 Grunt Shell

After invoking the Grunt shell, you can run your Pig scripts in the shell. In addition to that, there are certain useful shell and utility commands provided by the Grunt shell. This chapter explains the shell and utility commands provided by the Grunt shell.

Note – In some portions of this chapter, the commands like **Load** and **Store** are used. Refer the respective chapters to get in-detail information on them.

Shell Commands

The Grunt shell of Apache Pig is mainly used to write Pig Latin scripts. Prior to that, we can invoke any shell commands using **sh** and **fs**.

sh Command

Using **sh** command, we can invoke any shell commands from the Grunt shell. Using **sh** command from the Grunt shell, we cannot execute the commands that are a part of the shell environment (**ex** – cd).

Syntax

Given below is the syntax of **sh** command.

```
grunt> sh shell command parameters
```

Example

We can invoke the **ls** command of Linux shell from the Grunt shell using the **sh** option as shown below. In this example, it lists out the files in the **/pig/bin/** directory.

```
grunt> sh ls

pig
pig_1444799121955.log
pig.cmd
pig.py
```

fs Command

Using the **fs** command, we can invoke any FsShell commands from the Grunt shell.

Syntax

Given below is the syntax of **fs** command.

```
grunt> sh File System command parameters
```

Example

We can invoke the `ls` command of HDFS from the Grunt shell using `fs` command. In the following example, it lists the files in the HDFS root directory.

```
grunt> fs -ls
```

Found 3 items

```
drwxrwxrwx - Hadoop supergroup      0 2015-09-08 14:13 Hbase
drwxr-xr-x - Hadoop supergroup      0 2015-09-09 14:52 seqgen_data
drwxr-xr-x - Hadoop supergroup      0 2015-09-08 11:30 twitter_data
```

In the same way, we can invoke all the other file system shell commands from the Grunt shell using the `fs` command.

Utility Commands

The Grunt shell provides a set of utility commands. These include utility commands such as **clear**, **help**, **history**, **quit**, and **set**; and commands such as **exec**, **kill**, and **run** to control Pig from the Grunt shell. Given below is the description of the utility commands provided by the Grunt shell.

clear Command

The **clear** command is used to clear the screen of the Grunt shell.

Syntax

You can clear the screen of the grunt shell using the **clear** command as shown below.

```
grunt> clear
```

help Command

The **help** command gives you a list of Pig commands or Pig properties.

5.4 Pig Latin

Pig Latin is the language used to analyze data in Hadoop using Apache Pig. In this chapter, we are going to discuss the basics of Pig Latin such as Pig Latin statements, data types, general and relational operators, and Pig Latin UDF's.

Pig Latin – Data Model

As discussed in the previous chapters, the data model of Pig is fully nested. A **Relation** is the outermost structure of the Pig Latin data model. And it is a **bag** where –

- A bag is a collection of tuples.
- A tuple is an ordered set of fields.
- A field is a piece of data.

Pig Latin – Statemets

While processing data using Pig Latin, **statements** are the basic constructs.

- These statements work with **relations**. They include **expressions** and **schemas**.
- Every statement ends with a semicolon (;).
- We will perform various operations using operators provided by Pig Latin, through statements.
- Except LOAD and STORE, while performing all other operations, Pig Latin statements take a relation as input and produce another relation as output.
- As soon as you enter a **Load** statement in the Grunt shell, its semantic checking will be carried out. To see the contents of the schema, you need to use the **Dump** operator. Only after performing the **dump** operation, the MapReduce job for loading the data into the file system will be carried out.

Example

Given below is a Pig Latin statement, which loads data to Apache Pig.

```
grunt> Student_data = LOAD 'student_data.txt' USING PigStorage(',')as
( id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray );
```

5.5 Load and Store Operators

In general, Apache Pig works on top of Hadoop. It is an analytical tool that analyzes large datasets that exist in the **Hadoop File System**. To analyze data using Apache Pig, we have to initially load the data into Apache Pig. This chapter explains how to load data to Apache Pig from HDFS.

Preparing HDFS

In MapReduce mode, Pig reads (loads) data from HDFS and stores the results back in HDFS. Therefore, let us start HDFS and create the following sample data in HDFS.

| Student ID | First Name | Last Name | Phone | City |
|------------|------------|-------------|------------|--------------|
| 001 | Rajiv | Reddy | 9848022337 | Hyderabad |
| 002 | siddarth | Battacharya | 9848022338 | Kolkata |
| 003 | Rajesh | Khanna | 9848022339 | Delhi |
| 004 | Preethi | Agarwal | 9848022330 | Pune |
| 005 | Trupthi | Mohanthi | 9848022336 | Bhuwaneshwar |
| 006 | Archana | Mishra | 9848022335 | Chennai |

The above dataset contains personal details like id, first name, last name, phone number and city, of six students.

Step 1: Verifying Hadoop

First of all, verify the installation using Hadoop version command, as shown below.

```
$ hadoop version
```

If your system contains Hadoop, and if you have set the PATH variable, then you will get the following output –

```
Hadoop 2.6.0
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r
e3496499ecb8d220fba99dc5ed4c99c8f9e33bb1
Compiled by jenkins on 2014-11-13T21:10Z
Compiled with protoc 2.5.0
From source with checksum 18e43357c8f927c0695f1e9522859d6a
This command was run using /home/Hadoop/hadoop/share/hadoop/common/hadoop
common-2.6.0.jar
```

Step 2: Starting HDFS

Browse through the **sbin** directory of Hadoop and start **yarn** and Hadoop dfs (distributed file system) as shown below.

```
cd /$Hadoop_Home/sbin/
$ start-dfs.sh
localhost: starting namenode, logging to /home/Hadoop/hadoop/logs/hadoopHadoop-
namenode-localhost.localdomain.out
localhost: starting datanode, logging to /home/Hadoop/hadoop/logs/hadoopHadoop-datanode-
localhost.localdomain.out
Starting secondary namenodes [0.0.0.0]
starting secondarynamenode, logging to /home/Hadoop/hadoop/logs/hadoop-
Hadoopsecondarynamenode-localhost.localdomain.out

$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /home/Hadoop/hadoop/logs/yarn-
Hadoopresourcemanager-localhost.localdomain.out
localhost: starting nodemanager, logging to /home/Hadoop/hadoop/logs/yarnHadoop-
nodemanager-localhost.localdomain.out
```

Step 3: Create a Directory in HDFS

In Hadoop DFS, you can create directories using the command **mkdir**. Create a new directory in HDFS with the name **Pig_Data** in the required path as shown below.

```
$cd /$Hadoop_Home/bin/  
$ hdfs dfs -mkdir hdfs://localhost:9000/Pig_Data
```

Step 4: Placing the data in HDFS

The input file of Pig contains each tuple/record in individual lines. And the entities of the record are separated by a delimiter (In our example we used “,”).

In the local file system, create an input file **student_data.txt** containing data as shown below.

```
001,Rajiv,Reddy,9848022337,Hyderabad  
002,siddarth,Battacharya,9848022338,Kolkata  
003,Rajesh,Khanna,9848022339,Delhi  
004,Preethi,Agarwal,9848022330,Pune  
005,Trupthi,Mohanthi,9848022336,Bhuwaneshwar  
006,Archana,Mishra,9848022335,Chennai.
```

Now, move the file from the local file system to HDFS using **put** command as shown below. (You can use **copyFromLocal** command as well.)

```
$ cd $HADOOP_HOME/bin  
$ hdfs dfs -put /home/Hadoop/Pig/Pig_Data/student_data.txt hdfs://localhost:9000/pig_data/
```

Verifying the file

You can use the **cat** command to verify whether the file has been moved into the HDFS, as shown below.

```
$ cd $HADOOP_HOME/bin  
$ hdfs dfs -cat hdfs://localhost:9000/pig_data/student_data.txt
```

Output

You can see the content of the file as shown below.


```
15/10/01 12:16:55 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
```

```
001,Rajiv,Reddy,9848022337,Hyderabad
002,siddarth,Battacharya,9848022338,Kolkata
003,Rajesh,Khanna,9848022339,Delhi
004,Preethi,Agarwal,9848022330,Pune
005,Trupthi,Mohanthi,9848022336,Bhuwaneshwar
006,Archana,Mishra,9848022335,Chennai
```

The Load Operator

You can load data into Apache Pig from the file system (HDFS/ Local) using **LOAD** operator of **Pig Latin**.

Syntax

The load statement consists of two parts divided by the “=” operator. On the left-hand side, we need to mention the name of the relation **where** we want to store the data, and on the right-hand side, we have to define **how** we store the data. Given below is the syntax of the **Load** operator.

```
Relation_name = LOAD 'Input file path' USING function as schema;
```

Where,

- **relation_name** – We have to mention the relation in which we want to store the data.
- **Input file path** – We have to mention the HDFS directory where the file is stored. (In MapReduce mode)
- **function** – We have to choose a function from the set of load functions provided by Apache Pig (**BinStorage**, **JsonLoader**, **PigStorage**, **TextLoader**).
- **Schema** – We have to define the schema of the data. We can define the required schema as follows –

```
(column1 : data type, column2 : data type, column3 : data type);
```

Note – We load the data without specifying the schema. In that case, the columns will be addressed as \$01, \$02, etc... (check).

Storing Data

Assume we have a file **student_data.txt** in HDFS with the following content.

```
001,Rajiv,Reddy,9848022337,Hyderabad
002,siddarth,Battacharya,9848022338,Kolkata
003,Rajesh,Khanna,9848022339,Delhi
004,Preethi,Agarwal,9848022330,Pune
005,Trupthi,Mohanthi,9848022336,Bhuwaneshwar
006,Archana,Mishra,9848022335,Chennai.
```

And we have read it into a relation **student** using the LOAD operator as shown below.

```
grunt> student = LOAD 'hdfs://localhost:9000/pig_data/student_data.txt'
      USING PigStorage(',')
      as ( id:int, firstname:chararray, lastname:chararray, phone:chararray,
      city:chararray );
```

Now, let us store the relation in the HDFS directory “/pig_Output/” as shown below.

```
grunt> STORE student INTO 'hdfs://localhost:9000/pig_Output/' USING PigStorage(',');
```

5.6 Grouping and Joining

The **GROUP** operator is used to group the data in one or more relations. It collects the data having the same key.

Syntax

Given below is the syntax of the **group** operator.

```
grunt> Group_data = GROUP Relation_name BY age;
```

Grouping by Multiple Columns

Let us group the relation by age and city as shown below.

```
grunt> group_multiple = GROUP student_details by (age, city);
```

Group All

You can group a relation by all the columns as shown below.

```
grunt> group_all = GROUP student_details All;
```

The **JOIN** operator is used to combine records from two or more relations. While performing a join operation, we declare one (or a group of) tuple(s) from each relation, as keys. When these keys match, the two particular tuples are matched, else the records are dropped. Joins can be of the following types –

- Self-join
- Inner-join
- Outer-join – left join, right join, and full join

Self-join

Self-join is used to join a table with itself as if the table were two relations, temporarily renaming at least one relation.

Syntax

Given below is the syntax of performing **self-join** operation using the **JOIN** operator.

```
grunt> Relation3_name = JOIN Relation1_name BY key, Relation2_name BY key ;
```

Inner Join

Inner Join is used quite frequently; it is also referred to as **equijoin**. An inner join returns rows when there is a match in both tables.

It creates a new relation by combining column values of two relations (say A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, the column values for each matched pair of rows of A and B are combined into a result row.

Syntax

Here is the syntax of performing **inner join** operation using the **JOIN** operator.

```
grunt> result = JOIN relation1 BY columnname, relation2 BY columnname;
```

Left Outer Join

The **left outer Join** operation returns all rows from the left table, even if there are no matches in the right relation.

Syntax

Given below is the syntax of performing **left outer join** operation using the **JOIN** operator.

```
grunt> Relation3_name = JOIN Relation1_name BY id LEFT OUTER, Relation2_name BY customer_id;
```

5.7 Filtering

The **FILTER** operator is used to select the required tuples from a relation based on a condition.

Syntax

Given below is the syntax of the **FILTER** operator.

```
grunt> Relation2_name = FILTER Relation1_name BY (condition);
```

The **DISTINCT** operator is used to remove redundant (duplicate) tuples from a relation.

Syntax

Given below is the syntax of the **DISTINCT** operator.

```
grunt> Relation_name2 = DISTINCT Relatin_name1;
```

The **FOREACH** operator is used to generate specified data transformations based on the column data.

Syntax

Given below is the syntax of **FOREACH** operator.

```
grunt> Relation_name2 = FOREACH Relatin_name1 GENERATE (required data);
```

5.8 Sorting

The **ORDER BY** operator is used to display the contents of a relation in a sorted order based on one or more fields.

Syntax

Given below is the syntax of the **ORDER BY** operator.

```
grunt> Relation_name2 = ORDER Relatin_name1 BY (ASC|DESC);
```

The **LIMIT** operator is used to get a limited number of tuples from a relation.

Syntax

Given below is the syntax of the **LIMIT** operator.

```
grunt> Result = LIMIT Relation_name required number of tuples;
```

Chapter 6

APACHE HIVE

6.1 What is Hive?

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy. Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Hive is not

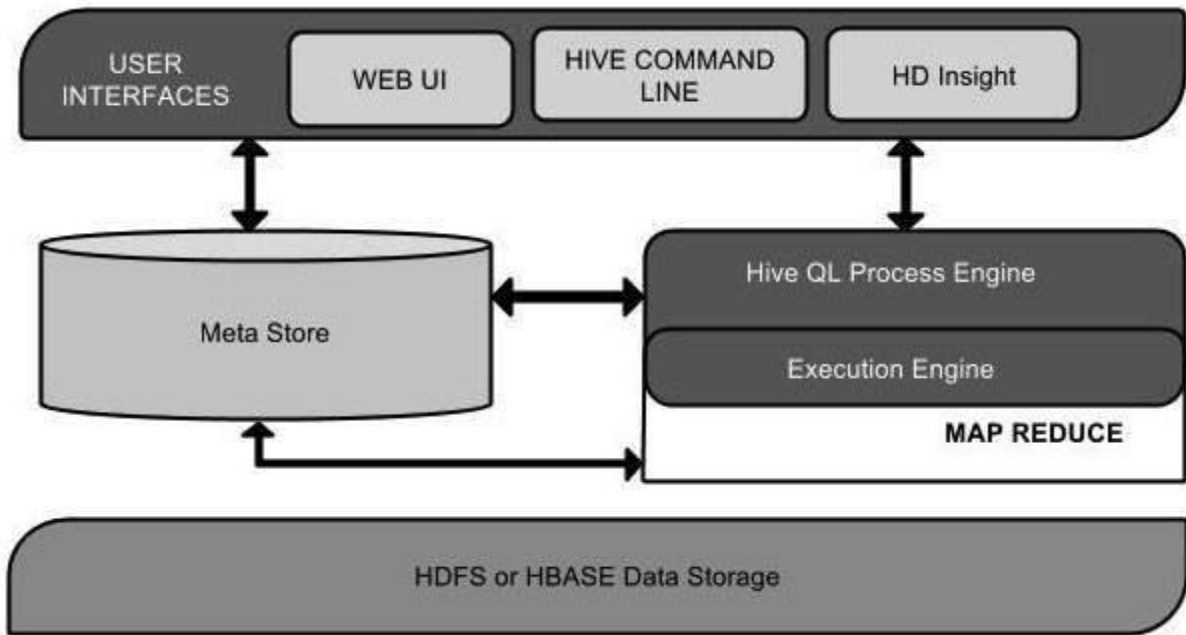
- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

Features of Hive

- It stores schema in a database and processed data into HDFS.
- It is designed for OLAP.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

Architecture of Hive

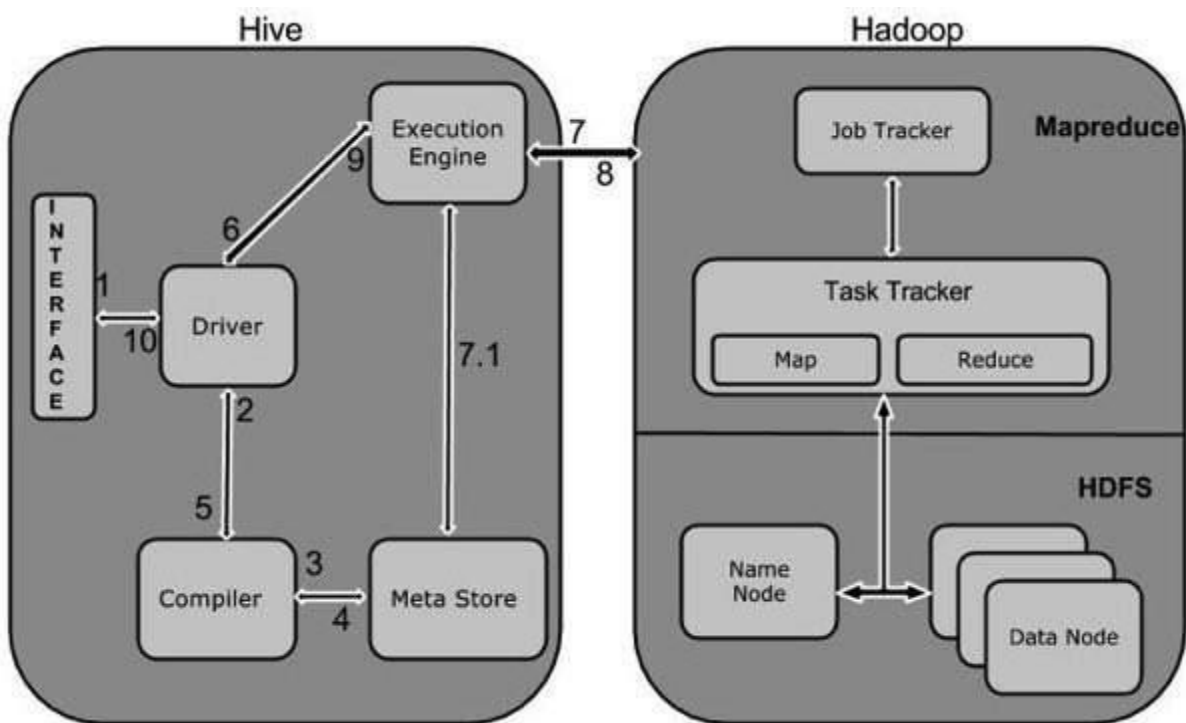
The following component diagram depicts the architecture of Hive:



This component diagram contains different units.

6.2 Working of Hive

The following diagram depicts the workflow between Hive and Hadoop.



The following table defines how Hive interacts with Hadoop framework:

| Step No. | Operation |
|----------|---|
| 1 | Execute Query The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute. |
| 2 | Get Plan The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query. |
| 3 | Get Metadata The compiler sends metadata request to Metastore (any database). |
| 4 | Send Metadata Metastore sends metadata as a response to the compiler. |
| 5 | Send Plan The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete. |
| 6 | Execute Plan The driver sends the execute plan to the execution engine. |
| 7 | Execute Job Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job. |
| 7.1 | Metadata Ops |

| | |
|----|--|
| | Meanwhile in execution, the execution engine can execute metadata operations with Metastore. |
| 8 | Fetch Result The execution engine receives the results from Data nodes. |
| 9 | Send Results The execution engine sends those resultant values to the driver. |
| 10 | Send Results The driver sends the results to Hive Interfaces. |

6.3 Data Types

All the data types in Hive are classified into four types, given as follows:

- Column Types
- Literals
- Null Values
- Complex Types

Column Types

Column type are used as column data types of Hive. They are as follows:

Integral Types

String Types

Timestamp

Dates

Decimals

Literals

The following literals are used in Hive:

Floating Point Types

Decimal Types

Null Value

Missing values are represented by the special value NULL.

Complex Types

The Hive complex data types are as follows:

Arrays

Maps

Structs

6.4 Hive Statements

Create Database Statement

Create Database is a statement used to create a database in Hive. A database in Hive is a **namespace** or a collection of tables. The **syntax** for this statement is as follows:

```
CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>
```

Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named **userdb**:

```
hive> CREATE DATABASE [IF NOT EXISTS] userdb;
```

or

```
hive> CREATE SCHEMA userdb;
```

The following query is used to verify a databases list:

```
hive> SHOW DATABASES;  
  
default  
  
userdb
```

Drop Database Statement

Drop Database is a statement that drops all the tables and deletes the database. Its syntax is as follows:

```
DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS] database_name  
[RESTRICT|CASCADE];
```

The following queries are used to drop a database. Let us assume that the database name is **userdb**.

```
hive> DROP DATABASE IF EXISTS userdb;
```

The following query drops the database using **CASCADE**. It means dropping respective tables before dropping the database.

```
hive> DROP DATABASE IF EXISTS userdb CASCADE;
```

The following query drops the database using **SCHEMA**.

```
hive> DROP SCHEMA userdb;
```

This clause was added in Hive 0.6.

Create Table Statement

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name
```

```
[(col_name data_type [COMMENT col_comment], ...)]
```

```
[COMMENT table_comment]
```

```
[ROW FORMAT row_format]
```

```
[STORED AS file_format]
```

Alter Table Statement

It is used to alter a table in Hive.

Syntax

The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

```
ALTER TABLE name RENAME TO new_name
```

```
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
```

```
ALTER TABLE name DROP [COLUMN] column_name
```

```
ALTER TABLE name CHANGE column_name new_name new_type
```

```
ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])
```

Rename To... Statement

The following query renames the table from **employee** to **emp**.

```
hive> ALTER TABLE employee RENAME TO emp;
```

Drop Table Statement

The syntax is as follows:

```
DROP TABLE [IF EXISTS] table_name;
```

The following query drops a table named **employee**:

```
hive> DROP TABLE IF EXISTS employee;
```

On successful execution of the query, you get to see the following response:

Partitioning

Hive organizes tables into partitions. It is a way of dividing a table into related parts based on the values of partitioned columns such as date, city, and department. Using partition, it is easy to query a portion of the data.

Tables or partitions are sub-divided into **buckets**, to provide extra structure to the data that may be used for more efficient querying. Bucketing works based on the value of hash function of some column of a table.

For example, a table named **Tab1** contains employee data such as id, name, dept, and yoj (i.e., year of joining). Suppose you need to retrieve the details of all employees who joined in 2012. A query searches the whole table for the required information. However, if you partition the employee data with the year and store it in a separate file, it reduces the query processing time. The following example shows how to partition a file and its data:

The following file contains employeedata table.

/tab1/employeedata/file1

```
id, name, dept, yoj
1, gopal, TP, 2012
2, kiran, HR, 2012
3, kaleel, SC, 2013
4, Prasanth, SC, 2013
```

The above data is partitioned into two files using year.

/tab1/employeedata/2012/file2

```
1, gopal, TP, 2012
2, kiran, HR, 2012
```

/tab1/employeedata/2013/file3

```
3, kaleel, SC, 2013
```

Adding a Partition

We can add partitions to a table by altering the table. Let us assume we have a table called **employee** with fields such as Id, Name, Salary, Designation, Dept, and yoj.

Syntax:

```
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec  
[LOCATION 'location1'] partition_spec [LOCATION 'location2'] ...;
```

partition_spec:

: (p_column = p_col_value, p_column = p_col_value, ...)

The following query is used to add a partition to the employee table.

```
hive> ALTER TABLE employee  
> ADD PARTITION (year='2013')  
> location '/2012/part2012';
```

Renaming a Partition

The syntax of this command is as follows.

```
ALTER TABLE table_name PARTITION partition_spec RENAME TO PARTITION  
partition_spec;
```

The following query is used to rename a partition:

```
hive> ALTER TABLE employee PARTITION (year='1203')  
> RENAME TO PARTITION (Yoj='1203');
```

Dropping a Partition

The following syntax is used to drop a partition:

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec, PARTITION
partition_spec,...;
```

The following query is used to drop a partition:

```
hive> ALTER TABLE employee DROP [IF EXISTS]
> PARTITION (year='1203');
```

6.5 Build in operators

There are four types of operators in Hive:

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Complex Operators

6.6 HiveQL

The Hive Query Language (HiveQL) is a query language for Hive to process and analyze structured data in a Metastore. This part explains how to use the SELECT statement with WHERE clause.

SELECT statement is used to retrieve the data from a table. WHERE clause works similar to a condition. It filters the data using the condition and gives you a finite result. The built-in operators and functions generate an expression, which fulfils the condition.

Syntax

Given below is the syntax of the SELECT query:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
```

```
[WHERE where_condition]

[GROUP BY col_list]

[HAVING having_condition]

[CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list]]

[LIMIT number];
```

The ORDER BY clause is used to retrieve the details based on one column and sort the result set by ascending or descending order.

Syntax

Given below is the syntax of the ORDER BY clause:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...

FROM table_reference

[WHERE where_condition]

[GROUP BY col_list]

[HAVING having_condition]

[ORDER BY col_list]]

[LIMIT number];
```

The GROUP BY clause is used to group all the records in a result set using a particular collection column. It is used to query a group of records.

Syntax

The syntax of GROUP BY clause is as follows:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...

FROM table_reference
```



```
[WHERE where_condition]
```

```
[GROUP BY col_list]
```

```
[HAVING having_condition]
```

```
[ORDER BY col_list]]
```

```
[LIMIT number];
```

JOIN is a clause that is used for combining specific fields from two tables by using values common to each one. It is used to combine records from two or more tables in the database. It is more or less similar to SQL JOIN.

Syntax

```
join_table:
```

```
table_reference JOIN table_factor [join_condition]
```

```
| table_reference { LEFT|RIGHT|FULL } [OUTER] JOIN table_reference
```

```
join_condition
```

```
| table_reference LEFT SEMI JOIN table_reference join_condition
```

```
| table_reference CROSS JOIN table_reference [join_condition]
```

Chapter 7

FINAL PROJECT

Sentiment Analysis on streaming Twitter data

From 20th century onwards this WWW has completely changed the way of expressing their views. Present situation is completely they are expressing their thoughts through online blogs, discussion forms and also some online applications like Facebook, Twitter, etc. If we take Twitter as our example nearly 1TB of text data is generating within a week in the form of tweets. So, by this it is understand clearly how this Internet is changing the way of living and style of people. Among these tweets can be categorized by the hash value tags for which they are commenting and posting their tweets. So, now many companies and also the survey companies are using this for doing some analytics such that they can predict the success rate of their product or also they can show the different view from the data that they have collected for analysis. But, to calculate their views is very difficult in a normal way by taking these heavy data that are going to generate day by day.



Apache Hadoop Ecosystem

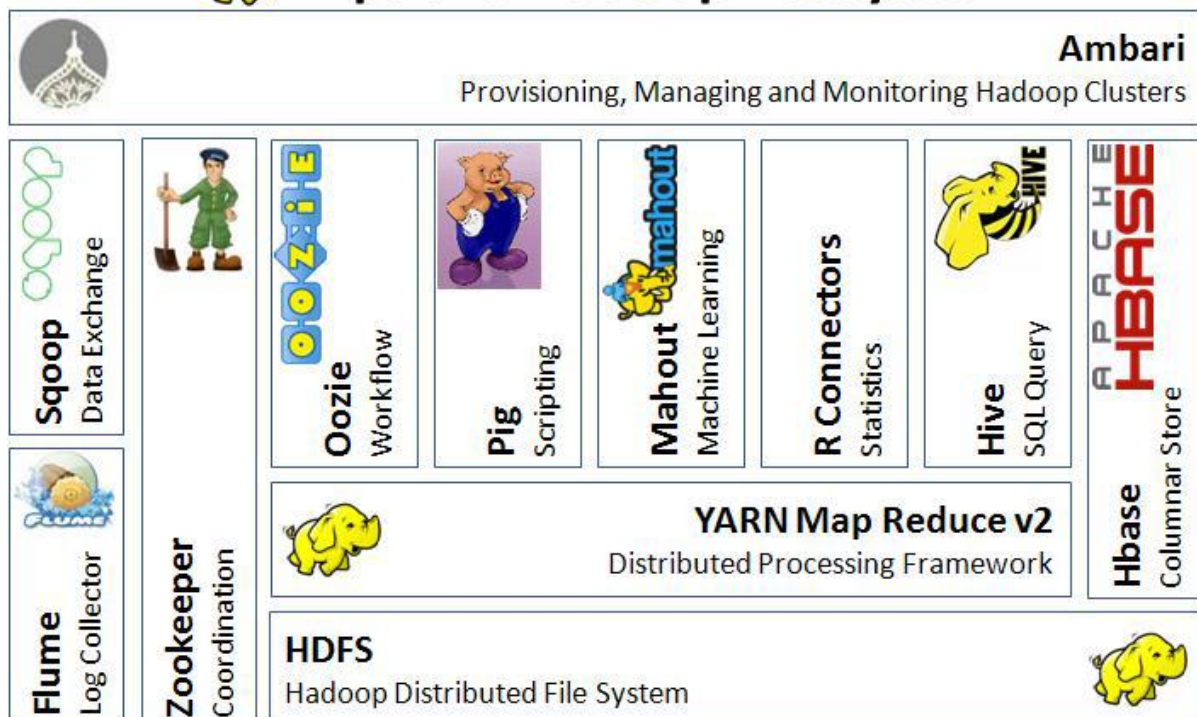


Fig. 1: Describes clearly Apache Hadoop Ecosystem.

The above figure shows clearly the different types of ecosystems that are available on Hadoop so, this problem is taking now and can be solved by using BIGDATA Problem as a solution. And if we consider getting the data from Twitter one should use any one programming language to crawl the data from their database or from their web pages. Coming to this problem here we are collecting this data by using BIGDATA online streaming Eco System Tool known as Flume and also the shuffling of data and generating them into structured data in the form of tables can be done by using Apache Hive.

7.1 PROBLEM STATEMENT

7.1.1 Existing System

As we have already discussed about the older way of getting data and also performing the sentiment analysis on those data. Here they are going to use some coding techniques for crawling the data from the twitter where they can extract the data from the Twitter web pages by using some code that may be written either in JAVA, Python etc. For those they are going to download the libraries that are provided by the twitter guys by using this they are crawling the data that we want particularly.

After getting raw data they will filter by using some old techniques and also they will find out the positive, negative and moderate words from the list of collected words in a text file. All these words should be collected by us to filter out or do some sentiment analysis on the filtered data. These words can be called as a dictionary set by which they will perform sentiment analysis. Also, after performing all these things and they want to store these in a database and coming to here they can use RDBMS[14] where they are having limitations in creating tables and also accessing the tables effectively.

7.1.2 Proposed System

As it can have seen existing system drawbacks, here we are going to overcome them by solving this issue using Big Data problem statement. So here we are going to use Hadoop and its Ecosystems, for getting raw data from the Twitter we are using Hadoop online streaming tool using Apache Flume. In this tool only we are going to configure everything that we want to get data from the Twitter. For this we want to set the configuration and also want to define what information that we want to get from Twitter. All these will be saved into our HDFS (Hadoop Distributed File System) in our prescribed format. From this raw data we are going to create the table and filter the information that is needed for us and sort them into the Hive Table. And

form that we are going to perform the Sentiment Analysis by using some UDF's (User Defined Functions) by which we can perform sentiment analysis by taking Stanford Core NLP[11] as the data dictionary so that by using that we can decide the list of words that coming under positive, moderate and negative.

The following figure shows clearly the architecture view for the proposed system by this we can understand how our project is effective using the Hadoop ecosystems and how the data is going to store form the Flume, also how it is going to create tables using Hive also how the sentiment analysis is going to perform.

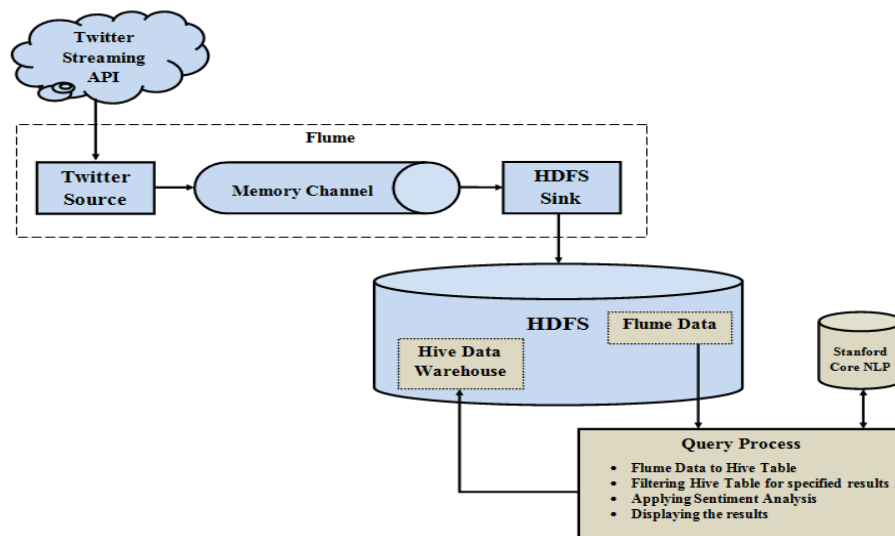
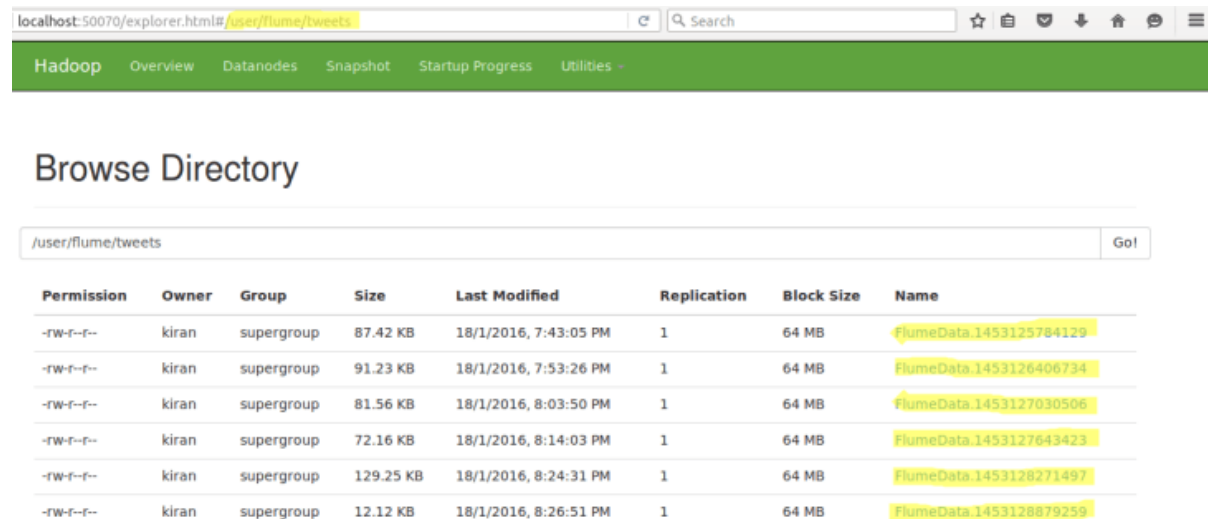


Fig. 2: Architecture diagram for proposed system.

7.2 METHODOLOGY

All the real-time tweets is kept in the location `‘/user/flume/tweets’` HDFS. You can refer to the below screen shot for the same.



| Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name |
|------------|-------|------------|-----------|-----------------------|-------------|------------|-------------------------|
| -rw-r--r-- | kiran | supergroup | 87.42 KB | 18/1/2016, 7:43:05 PM | 1 | 64 MB | FlumeData.1453125784129 |
| -rw-r--r-- | kiran | supergroup | 91.23 KB | 18/1/2016, 7:53:26 PM | 1 | 64 MB | FlumeData.1453126406734 |
| -rw-r--r-- | kiran | supergroup | 81.56 KB | 18/1/2016, 8:03:50 PM | 1 | 64 MB | FlumeData.1453127030506 |
| -rw-r--r-- | kiran | supergroup | 72.16 KB | 18/1/2016, 8:14:03 PM | 1 | 64 MB | FlumeData.1453127643423 |
| -rw-r--r-- | kiran | supergroup | 129.25 KB | 18/1/2016, 8:24:31 PM | 1 | 64 MB | FlumeData.1453128271497 |
| -rw-r--r-- | kiran | supergroup | 12.12 KB | 18/1/2016, 8:26:51 PM | 1 | 64 MB | FlumeData.1453128879259 |

The data from Twitter is in ‘Json’ format, so a Pig JsonLoader is required to load the data into Pig.

```
REGISTER '/home/kiran/Desktop/elephant-bird-hadoop-compat-4.1.jar';
```

```
REGISTER '/home/kiran/Desktop/elephant-bird-pig-4.1.jar';
```

```
REGISTER '/home/kiran/Desktop/json-simple-1.1.1.jar';
```

After registering the required jars, we can now write a Pig script to perform Sentiment Analysis.

Below is a sample tweets collected for this purpose:

```
{ "filter_level": "low", "retweeted": false, "in_reply_to_screen_name": "FilmFan", "truncated":  
false, "lang": "en", "in_reply_to_status_id_str": null, "id": "689085590822891521", "in_reply_to  
_user_id_str": "6048122", "timestamp_ms": "1453125782100", "in_reply_to_status_id": null,  
1 "created_at": "Mon Jan 18 14:03:02 +0000 2016", "favorite_count": 0, "place": null, "coordina  
tes": null, "text": "@filmfan hey its time for you guys follow @acadgild To #AchieveMore a  
nd participate in contest Win Rs.500 worth vouchers", "contributors": null, "geo": null, "entiti
```

```

es":{"symbols":[],"urls":[],"hashtags":[{"text":"AchieveMore","indices":[56,68]}],"user_
mentions":[{"id":"6048122","name":"Tanya","indices":[0,8],"screen_name":"FilmFan","id_
str":"6048122"},{"id":"2649945906","name":"ACADGILD","indices":[42,51],"screen_nam
e":"acadgild","id_str":"2649945906"}]},{"is_quote_status":false,"source":"<a href=\"https:/
/about.twitter.com/products/tweetdeck\" rel=\"nofollow\">TweetDeck</a>","favorited":fa
lse,"in_reply_to_user_id":"6048122","retweet_count":0,"id_str":"689085590822891521","u
ser":{"location":"India","default_profile":false,"profile_background_tile":false,"statuses_
count":86548,"lang":"en","profile_link_color":"94D487","profile_banner_url":"https://pbs
.twimg.com/profile_banners/197865769/1436198000","id":197865769,"following":null,"p
rotected":false,"favourites_count":1002,"profile_text_color":"000000","verified":false,"de
scription":"Proud Indian, Digital Marketing Consultant,Traveler, Foodie, Adventurer, Data
Architect, Movie Lover, Namo Fan","contributors_enabled":false,"profile_sidebar_border_
color":"000000","name":"Bahubali","profile_background_color":"000000","created_at":"
Sat Oct 02 17:41:02 +0000 2010","default_profile_image":false,"followers_count":4467,"p
rofile_image_url_https":"https://pbs.twimg.com/profile_images/664486535040000000/GO
jDUiuK_normal.jpg","geo_enabled":true,"profile_background_image_url":"http://abs.twi
mg.com/images/themes/theme1/bg.png","profile_background_image_url_https":"https://a
bs.twimg.com/images/themes/theme1/bg.png","follow_request_sent":null,"url":null,"utc_o
ffset":19800,"time_zone":"Chennai","notifications":null,"profile_use_background_image"
:false,"friends_count":810,"profile_sidebar_fill_color":"000000","screen_name":"Ashok_
Uppuluri","id_str":"197865769","profile_image_url":"http://pbs.twimg.com/profile_image
s/664486535040000000/GOjDUiuK_normal.jpg","listed_count":50,"is_translator":false}}

```

The tweets are in nested Json format and consists of map data types. We need to load the tweets using JsonLoader which supports maps, so we are using **elephant bird JsonLoader** to load the tweets.

Below is the first Pig statement required to load the tweets into Pig:

```

load_tweets = LOAD '/user/flume/tweets/' USING com.twitter.elephantbird.pig.load.JsonL
oader('-nestedLoad') AS myMap;

```



```

grunt> load_tweets = LOAD '/user/flume/tweets/' USING com.twitter.elphatbird.pig.load.JsonLoader('nestedLoad') AS myMap;
2016-01-20 16:25:06,415 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2016-01-20 16:25:06,433 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning USING_OVERLOADED_FUNCTION 9 time(s).
2016-01-20 16:25:06,433 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 9 time(s).
2016-01-20 16:25:06,433 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 8 time(s).
2016-01-20 16:25:06,433 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_MAP 33 time(s).
2016-01-20 16:25:06,433 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_LONG 14 time(s).
grunt>

```

When we dump the above relation, we can see that all the tweets got loaded successfully.

```

[[filter_level#low,text#Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/yOvpz2ov4q,contributors
#geo#,retweeted#false,in_reply_to_screen_name#,possibly_sensitive#false,truncated#false,lang#en,entities#{hashtags:[{text#telcos,indices#(19),(
26)}]},[text#Hadoop,indices#(42),(49)}]}, symbols=[], urls=[{display_url#lbn.co/1KhvqXJ,expanded_url#http://lbn.co/1KhvqXJ,indices#(85),(16
9)},url#https://t.co/yOvpz2ov4q}], user_mentions=[],in_reply_to_status_id_str#,is_quote_status#false,id#689096012196085760,source#<a href="http
://twitter.com/download/android" rel="nofollow">Twitter For Android</a>,in_reply_to_user_id_str#,favorited#false,timestamp_ms#1453128266749,in_re
ply_to_status_id#,retweet_count#0,in_reply_to_user_id#,created_at#Mon Jan 18 14:44:26 +0000 2016,favorite_count#0,id_str#689096012196085760,place
#,user#{location#Commonwealth of Massachusetts,default_profile#false,statuses_count#22991,profile_background_tile#true,lang#en,profile_link
_color#190069,profile_banner_url#https://pbs.twimg.com/profile_banners/237413764/1449519792,id#237413764,following#null,favourites_count#7992,
protected#false,profile_text_color#000000,contributors_enabled#false,description#Content Marketing @IBMAnalytics.Father to the #AdventureMen
Cheshire YMCA Board Member. Volunteer @CampTakodah. Chaplain of Trinity Lodge AF&AM. Loud speaker.,verified#false,name#J. Graene Noseworthy,fol
lowers_count#3153,geo_enabled#true,profile_image_url_https#https://pbs.twimg.com/profile_images/686880402871562242/Lxn73Ql1_normal.jpg,profile
_background_image_url#http://pbs.twimg.com/profile_background_images/674975457109088384/EXfxvcJ8.jpg,profile_background_image_url_https#https://
pbs.twimg.com/profile_background_images/674975457109088384/EXfxvcJ8.jpg,follow_request_sent#null,url#http://linkd.in/bDH7pI,utc_offset#-18000,
time_zone#Eastern Time (US & Canada),notifications#null,friends_count#2542,profile_use_background_image#true,profile_sidebar_fill_color#0000
00,screen_name#graeneknows,id_str#237413764,profile_image_url#http://pbs.twimg.com/profile_images/686880402871562242/Lxn73Ql1_normal.jpg,is_t
ranslator#false,listed_count#404,coordinates#)}
[[filter_level#low,retweeted#false,in_reply_to_screen_name#,possibly_sensitive#false,truncated#false,lang#en,in_reply_to_status_id_str#,id#689096
024854523907,extended_entities#{media:[{id#689096024632225792,sizes#{small#{w#340,h#167,resize#fit},thumb#{w#150,h#150,resize#crop},medium
#{w#600,h#295,resize#fit},large#{w#640,h#315,resize#fit}},media_url_https#https://pbs.twimg.com/media/CZApLvRWcAAIexa.png,media_url#http://p
bs.twimg.com/media/CZApLvRWcAAIexa.png,expanded_url#http://twitter.com/Tallen_BigData/status/689096024854523907/photo/1,indices#(96),(119)},id_s
tr#689096024632225792,display_url#pic.twitter.com/C9Xx3nUHRi,type#photo,url#https://t.co/C9Xx3nUHRi}],in_reply_to_user_id_str#,timestamp_ms#145
3128269767,in_reply_to_status_id#,created_at#Mon Jan 18 14:44:29 +0000 2016,favorite_count#0,place#,coordinates#,text#Got #bigdata? Manage it on
your own terms with #Hadoop and @SASDataMGMT https://t.co/xPNW7jUn4F https://t.co/C9Xx3nUHRi,contributors#geo#,entities#{hashtags:[{text#bigdat
a,indices#(4),(12)}]},[text#Hadoop,indices#(47),(54)}]}, symbols=[], media:[{id#689096024632225792,sizes#{small#{w#340,h#167,resize#fit},
thumb#{w#150,h#150,resize#crop},medium#{w#600,h#295,resize#fit},large#{w#640,h#315,resize#fit}},media_url_https#https://pbs.twimg.com/me
dia/CZApLvRWcAAIexa.png,media_url#http://pbs.twimg.com/media/CZApLvRWcAAIexa.png,expanded_url#http://twitter.com/Tallen_BigData/status/68909602485
4523907/photo/1,indices#(96),(119)},id_str#689096024632225792,display_url#pic.twitter.com/C9Xx3nUHRi,type#photo,url#https://t.co/C9Xx3nUHRi}],
urls=[{display_url#bit.ly/1nekW08,expanded_url#http://bit.ly/1nekW08,indices#(72),(95)},url#https://t.co/xPNW7jUn4F}], user_mentions=[{id#266
93930,indices#(59),(71)},screen_name#SASDataMGMT,id_str#26093930,name#SAS Data Management}],is_quote_status#false,source#<a href="http://login
.voicestorm.com" rel="nofollow">VoiceStorm</a>,favorited#false,retweet_count#0,in_reply_to_user_id#,id_str#689096024854523907,user#{location#null,
default_profile#true,statuses_count#407,profile_background_tile#false,lang#en,profile_link_color#0084B4,profile_banner_url#https://pbs.twi
ng.com/profile_banners/3075179092/1425667328,id#3075179092,following#null,favourites_count#1,protected#false,profile_text_color#333333,cont
ributors_enabled#false,description#null,verified#false,name#Taylor Allen,profile_sidebar_border_color#C0DEED,profile_background_color#C0DEED
,created_at#Fri Mar 06 16:11:55 +0000 2015,default_profile_image#false,followers_count#19,geo_enabled#false,profile_image_url_https#https://
pbs.twimg.com/profile_images/573879073904029696/h8c_3mg2_normal.jpeg,profile_background_image_url#http://abs.twimg.com/images/themes/theme1/bg.p
ng,profile_background_image_url_https#https://abs.twimg.com/images/themes/theme1/bg.png,follow_request_sent#null,url#null,utc_offset#-18000,

```

Now, we shall extract the **id** and the **tweet text** from the above tweets. The Pig statement necessary to perform this is as shown below:

```

1  extract_details = FOREACH load_tweets GENERATE myMap#'id' as id,myMap#'text' as t
   ext;

```

```

grunt> extract_details = FOREACH load_tweets GENERATE myMap#'id' as id,myMap#'text' as text;
2016-01-20 16:45:43,552 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning USING_OVERLOADED_FUNCTION 9 time(s).
2016-01-20 16:45:43,552 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 9 time(s).
2016-01-20 16:45:43,552 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 8 time(s).
2016-01-20 16:45:43,552 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_MAP 35 time(s).
2016-01-20 16:45:43,552 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_LONG 14 time(s).

```

We can see the extracted **id** and **tweet text** from the tweets in the below screen shot.

```
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q)
(689096024854523907,Got #bigdata? Manage it on your own terms with #Hadoop and @SASDataMGMT https://t.co/xPNW7JUm4F https://t.co/C9Xx3nUuRI)
(689096247324610560,Hadoop & Big data Trainers https://t.co/ed2JN3B0fP #DDA)
(689096260626554240,Big Data: Success Stories And Trends Beyond Hadoop https://t.co/aEk2CAu6gn #DDA)
(689096309018615809,Cloudera Hue & Apache Ambari
@gethue
@ApacheAmbari
#hadoop
#bigdata
#hue
#ambari)
(689096311740755968,RT @MobinRanjbar: Cloudera Hue & Apache Ambari
@gethue
@ApacheAmbari
#hadoop
#bigdata
#hue
#ambari)
(689096342866653184,All you need to know about Hadoop https://t.co/SXnKoN9yTc)
(689096424814997504,Blend, munge, and prep your #Hadoop #data faster w/ #spark and #impala https://t.co/4F50Yk7j8U https://t.co/PR7Zm6jhpS)
(689096488622895105,#Infonotics #InformationGovernance #dataquality #chiefdataofficer #Hadoop #masterdata all at #GartnerEIM #GartnerMDM https://t.co/2CQVcT5k95)
(689096509455994880,Disruptive Possibilities: How Big Data Changes Everything https://t.co/WUVDv1J9jV #DataScience #Hadoop)
(689096550790860801,Webinar with @SAS and @Cloudera, Jan 21 11am EST - Insurers Capitalize on #BigData #Analytics and #Hadoop https://t.co/GCjn7EwMko)
(689096804399404929,RT @Tallen_BigData: Got #bigdata? Manage it on your own terms with #Hadoop and @SASDataMGMT https://t.co/xPNW7JUm4F https://t.co/C9Xx3nUuRI)
(68909686966390784,#BigData: Success Stories And Trends Beyond #Hadoop https://t.co/dutZspFPfZ)
(689096849907683328,#Infonotics #InformationGovernance #dataquality #chiefdataofficer #Hadoop #masterdata at #GartnerEIM #GartnerMDM https://t.co/AnjpyVyDlq)
(689096862750629888,RT @analyticbridge: All you need to know about Hadoop https://t.co/SXnKoN9yTc)
(689096960209457155,SAS Grid Manager for #Hadoop nicely tied into YARN (Part 1) https://t.co/UthlSdeNvO)
(689096967968964608,RT @codespano: Why building an enterprise #data strategy https://t.co/nLiurnaI4l #Hadoop #bigdata)
(689096982749667330,tunguz: Disruptive Possibilities: How Big Data Changes Everything https://t.co/MyRRZuQRJU #DataScience #Hadoop)
(68909721175645184,Speed data management processes on #spark with SAS Data Loader for #Hadoop #newrelease. Download free trial now! https://t.co/LrcPEirMVL)
(689097219094515713,RT @analyticbridge: All you need to know about Hadoop https://t.co/SXnKoN9yTc)
(689097262383935491,Blend, munge, and prep your #data faster using #spark and #impala with SAS Data Loader for #Hadoop #newrelease https://t.co/2
```

We have the tweet id and the tweet text in the relation named as **extract_details**. Now, we shall extract the words from the text using the TOKENIZE key word in Pig.

1 tokens = foreach extract_details generate id,text, FLATTEN(TOKENIZE(text)) As word;

```
grunt> tokens = foreach extract_details generate id,text, FLATTEN(TOKENIZE(text)) As word;
2016-01-20 16:51:10,915 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning USING_OVERLOADED_FUNCTION 10 time(s).
2016-01-20 16:51:10,916 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 10 time(s).
2016-01-20 16:51:10,916 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 8 time(s).
2016-01-20 16:51:10,916 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_MAP 35 time(s).
2016-01-20 16:51:10,916 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_LONG 14 time(s).
grunt>
```

From the below screen shot, we can see that the text got divided into words.

```
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,[Podcast])
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,Hear)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,how)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,#telcos)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,can)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,use)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,Apache)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,#Hadoop)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,to)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,keep)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,up)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,with)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,rapid)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,data)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,growth:)
(689096012196085760,[Podcast] Hear how #telcos can use Apache #Hadoop to keep up with rapid data growth: https://t.co/y0vpz2ov4q,https://t.co/y0vpz2ov4q)
```


Now, we have to analyse the Sentiment for the tweet by using the words in the text. We will rate the word as per its meaning from +5 to -5 using the dictionary AFINN. The AFINN is a dictionary which consists of 2500 words which are rated from +5 to -5 depending on their meaning. You can download the dictionary from the following link:

[AFINN dictionary](#)

We will load the dictionary into pig by using the below statement:

```
1 dictionary = load '/AFINN.txt' using PigStorage('\t') AS(word:chararray,rating:int);
```

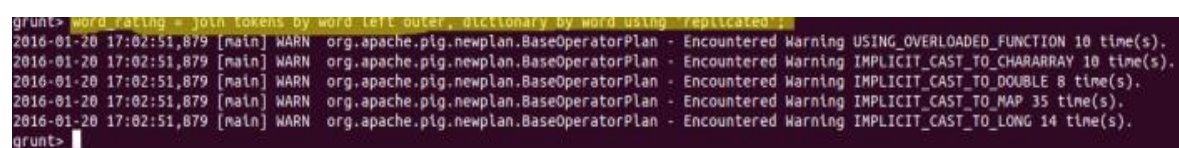
We can see the contents of the AFINN dictionary in the below screen shot.



```
(tricked,-2)
(trickery,-2)
(triumph,4)
(triumphant,4)
(trouble,-2)
(troubled,-2)
(troubles,-2)
(true,2)
(trust,1)
(trusted,2)
(tumor,-2)
(twat,-5)
(ugly,-3)
(unacceptable,-2)
(unappreciated,-2)
(unapproved,-2)
(unaware,-2)
(unbelievable,-1)
(unbelieving,-1)
(unbiased,2)
(uncertain,-1)
(unclear,-1)
(uncomfortable,-2)
(unconcerned,-2)
(unconfirmed,-1)
(unconvinced,-1)
(uncredited,-1)
(undecided,-1)
(underestimate,-1)
(underestimated,-1)
(underestimates,-1)
(underestimating,-1)
(undermine,-2)
(undermined,-2)
(undermines,-2)
(undermining,-2)
(undeserving,-2)
(undesirable,-2)
(uneasy,-2)
(unemployment,-2)
(unequal,-1)
```

Now, let's perform a map side join by joining the **tokens** statement and the dictionary contents using this command:

```
1 word_rating = join tokens by word left outer, dictionary by word using 'replicated';
```



```
grunt> word_rating = join tokens by word left outer, dictionary by word using 'replicated';
2016-01-20 17:02:51,879 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning USING_OVERLOADED_FUNCTION 10 time(s).
2016-01-20 17:02:51,879 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 10 time(s).
2016-01-20 17:02:51,879 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 8 time(s).
2016-01-20 17:02:51,879 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_MAP 35 time(s).
2016-01-20 17:02:51,879 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_LONG 14 time(s).
grunt>
```

We can see the schema of the statement after performing join operation by using the below command:

```
1 describe word_rating;
```

```
grunt> describe word_rating;
2016-01-20 17:06:11,823 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning USING_OVERLOADED_FUNCTION 1 time(s).
2016-01-20 17:06:11,823 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 1 time(s).
2016-01-20 17:06:11,823 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_MAP 2 time(s).
word_rating: {tokens::id: bytearray,tokens::text: bytearray,tokens::word: chararray,dictionary::word: chararray,dictionary::rating: int}
grunt>
```

In the above screenshot, we can see that the word_rating has joined the **tokens**(consists of id, tweet text, word) statement and the **dictionary**(consists of word, rating). Now we will extract the **id,tweet text** and **word rating**(from the dictionary) by using the below relation:

```
rating = foreach word_rating generate tokens::id as id,tokens::text as text, dictionary::rating
1 g as rate;
```

```
grunt> rating = foreach word_rating generate tokens::id as id,tokens::text as text, dictionary::rating as rate;
2016-01-20 17:12:10,655 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning USING_OVERLOADED_FUNCTION 10 time(s).
2016-01-20 17:12:10,655 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 10 time(s).
2016-01-20 17:12:10,655 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 8 time(s).
2016-01-20 17:12:10,655 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_MAP 35 time(s).
2016-01-20 17:12:10,655 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_LONG 14 time(s).
grunt>
```

We can now see the schema of the relation **rating** by using the command describe rating.

```
grunt> describe rating
2016-01-20 17:14:50,870 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning USING_OVERLOADED_FUNCTION 1 time(s).
2016-01-20 17:14:50,871 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 1 time(s).
2016-01-20 17:14:50,871 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_MAP 2 time(s).
rating: {id: bytearray,text: bytearray,rate: int}
grunt>
```

In the above screen shot we can see that our relation now consists of **id,tweet text** and **rate**(for each word). Now, we will group the **rating of all the words in a tweet** by using the below relation:

```
1 word_group = group rating by (id,text);
```

```

grunt> word_group = group rating by (id,text);
2016-01-20 17:17:26,982 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning USING_OVERLOADED_FUNCTION 10 time(s).
2016-01-20 17:17:26,982 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 10 time(s).
2016-01-20 17:17:26,982 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 8 time(s).
2016-01-20 17:17:26,982 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_MAP 35 time(s).
2016-01-20 17:17:26,982 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_LONG 14 time(s).
grunt>

```

Here we have grouped by two constraints, **id** and **tweet text**. Now, let's perform the **Average** operation on the **rating of the words per each tweet**.

avg_rate = foreach word_group generate group, AVG(rating.rate) as tweet_rating;

```

grunt> avg_rate = foreach word_group generate group, AVG(rating.rate) as tweet_rating;
2016-01-20 17:22:23,785 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning USING_OVERLOADED_FUNCTION 10 time(s).
2016-01-20 17:22:23,785 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 10 time(s).
2016-01-20 17:22:23,785 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 8 time(s).
2016-01-20 17:22:23,785 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_MAP 35 time(s).
2016-01-20 17:22:23,785 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_LONG 14 time(s).
grunt>

```

Now we have calculated the **Average rating of the tweet using the rating of the each word**. You can refer to the below image for the same.

```

(689085590822891521,@filmfan hey its time for you guys follow @acdgild To #AchieveMore and participate in contest Win Rs.500 worth vouchers),4.0)
(689085611639205888,@sujitjohn Follow @acdgild To #AchieveMore & participate in contest Hurry up! & win Flipkart vouchers),4.0)
(689085636456923138,@sujitalwani Hey tweeps want to win Flipkart vouchers so follow @acdgild To #AchieveMore contest Prizes of Rs.500),2.5)
(689085663334035456,@ln_bicky Hey Friends & Tweeps Please Follow @acdgild To #AchieveMore and participate in contest Be lucky get Rs.500 vouchers),3.0)
(689085686390863104,@ShreyVithalani You don't wanna miss this Go follow @acdgild To #AchieveMore and play in contest & win Flipkart vouchers),1.0)
(689085696619974656,RT @codespano: Why building an enterprise #data strategy https://t.co/nLiurnaI4l #Hadoop #bigdata),)
(689085710079537154,@SANGEETAAGRAMA Tweethearts! Follow @acdgild To #AchieveMore and participate in contest and win Flipkart vouchers! Aye!),4.0)
(689085731420131328,Urgent Need: Hadoop Developer_Sunnyvale, CA_6+ Months https://t.co/s3Hq0Jmj0R Need: Hadoop Developer_Sunnyvale, CA_6+ Months),)
(689085740374949888,@ltzzmesush Lets make ur day Fantastic! Follow @acdgild To #AchieveMore and participate in contest Prizes of Rs.500),)
(689085765154897920,Weblogic Admin with Oracle Strong-MI & BA Hadoop, Teradata + healthcare domain-MN, NJ, CT https://t.co/Uy0t5B5P1n https://t.co/Pm8VawKU17),)
(689085776731213824,@AryanSarath Contest freaks! Follow @acdgild To #AchieveMore and play in contest & win shopping vouchers Wohooo!),4.0)

```

From the above relation, we will get all the tweets i.e., both positive and negative.

Here, we can classify the positive tweets by taking the rating of the tweet which can be from **0-5**. We can classify the negative tweets by taking the rating of the tweet from **-5 to -1**.

We have now successfully performed the Sentiment Analysis on Twitter data using Pig. We now have the tweets and its rating, so let's perform an operation to filter out the positive tweets.

Now we will filter the positive tweets using the below statement:

1 positive_tweets = filter avg_rate by tweet_rating>=0;

```

grunt> positive_tweets = filter_avg_rate_by_tweet_rating>=0;
2016-01-20 17:28:47,087 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning USING_OVERLOADED_FUNCTION 10 time(s).
2016-01-20 17:28:47,087 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_CHARARRAY 10 time(s).
2016-01-20 17:28:47,087 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 9 time(s).
2016-01-20 17:28:47,087 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_MAP 35 time(s).
2016-01-20 17:28:47,087 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_LONG 14 time(s).
grunt>

```

We can see the positive tweets and its rating in the below screen shot.

```

((68908501139205880,@sujitjohn Follow @acadgild To #AchieveMore & participate in contest Hurry up! & win Flipkart vouchers),4.0)
((689085636456923138,@sujitlalwani Hey tweeps want to win Flipkart vouchers so follow @acadgild To #AchieveMore contest Prizes of Rs.500),2.5)
((689085663334035456,@in_bicky Hey Friends & Tweeps Please Follow @acadgild To #AchieveMore and participate in contest Be lucky get Rs.500 vouchers),3.0)
((689085686390063104,@ShreyVithalani You don't wanna miss this Go Follow @acadgild To #AchieveMore and play in contest & win Flipkart vouchers),1.0)
((689085710079537154,@SANGEETAAGRAWA Tweethearts! Follow @acadgild To #AchieveMore and participate in contest and win Flipkart vouchers! Aye!),4.0)
((689085776731213824,@AryanSarath Contest freaks! Follow @acadgild To #AchieveMore and play in contest & win shopping vouchers Wohooo!),4.0)
((689085820674945025,@RaviThakkar Its time to follow @acadgild To #AchieveMore and participate in contest You can win shopping vouchers),4.0)
((689085845538762752,@Iwant_us Follow @acadgild To #AchieveMore and participate in contest & Win Flipkart vouchers! Win Rs.500 worth vouchers),2.0)
((689085871522496518,@DeEp_ Its Crackling! The Contest will blow your mind! follow @acadgild To #AchieveMore & win Flipkart vouchers),4.0)
((689085896730263553,@sutharsweta MASSIVE contest FOLLOW @acadgild To #AchieveMore and play in contest & win Flipkart vouchers Win Rs.500 worth vouchers),3.0)
((689085921854124032,@ygeshni Tweethearts!! Follow @acadgild To #AchieveMore and participate in contest Hurry Up! & win Flipkart vouchers),4.0)
((689085951352680448,@adi_shah Tweethearts! Wanna beat your day blues! Follow @acadgild To #AchieveMore & play in contest & win Flipkart vouchers),4.0)
((689087505174540288,If you want to have a simple explanation of #HDFS works and you like cartoons, this is the place: https://t.co/WQqkx7QpP3 #Hadoop #BigData),1.5)
((689088647493189632,RT @rick_vanderlans: If you want to have a simple explanation of #HDFS works and you like cartoons, this is the place: https://t.co/WQqkx7Q...),1.5)
((689089909542531072,RT @rick_vanderlans: If you want to have a simple explanation of #HDFS works and you like cartoons, this is the place: https://t.co/WQqkx7Q...),1.5)
((689089951296827392,RT @rick_vanderlans: If you want to have a simple explanation of #HDFS works and you like cartoons, this is the place: https://t.co/WQqkx7Q...),1.5)
((689090948213014528,Speed data management processes on #spark with SAS Data Loader for #Hadoop #newrelease. Download free trial now! https://t.co/H4YauGjCr),1.0)

```

In the above screen shot we can see the tweet_id,tweet_text and its rating.

CONCLUSION

There are different ways to get Twitter data or any other online streaming data where they want to code lines of coding to achieve this. And, also they want to perform the sentiment analysis on the stored data where it makes some complex to perform those operations. Coming to this project we have achieved by this problem statement and solving it in BIG DATA by using Hadoop and its EcoSystems. And finally we have done sentiment analysis on the Twitter data that is stored in HDFS. So, here the processing time taken is also very less compared to the previous methods because Apache Flume and Pig are the best methods to process large amount of data in a small time.

The availability of Big Data, low-cost commodity hardware, and new information management and analytic software have produced a unique moment in the history of data analysis. The convergence of these trends means that we have the capabilities required to analyze astonishing data sets quickly and cost-effectively for the first time in history. These capabilities are neither theoretical nor trivial. They represent a genuine leap forward and a clear opportunity to realize enormous gains in terms of efficiency, productivity, revenue, and profitability. The Age of Big Data is here, and these are truly revolutionary times if both business and technology professionals continue to work together and deliver on the promise.

LEARNINGS

1. Big Data and its evolution
2. 5 important V's of Big Data
3. Hadoop Distributed File System
4. Hadoop Clusters
5. Hadoop MapReduce using python 3.6
6. Hive
7. Flume
8. Pig
9. Linux
10. Twitter API

APPENDICES

Cloudera's Distribution Including Apache Hadoop

Cloudera's Distribution Including Apache Hadoop (hereafter *CDH*) is an integrated Apache Hadoop-based stack containing all the components needed for production, tested and packaged to work together. Cloudera makes the distribution available in a number of different formats: Linux packages, virtual machine images, tarballs, and tools for running CDH in the cloud. CDH is free, released under the Apache 2.0 license, and available at <http://www.cloudera.com/cdh>.

As of CDH 5, the following components are included, many of which are covered elsewhere in this book:

Apache Avro

A cross-language data serialization library; includes rich data structures, a fast/compact binary format, and RPC

Apache Crunch

A high-level Java API for writing data processing pipelines that can run on MapReduce or Spark

Apache DataFu (incubating)

A library of useful statistical UDFs for doing large-scale analyses

Apache Flume

Highly reliable, configurable streaming data collection

Apache Hadoop

Highly scalable data storage (HDFS), resource management (YARN), and processing (MapReduce)

Apache HBase

Column-oriented real-time database for random read/write access

Apache Hive

SQL-like queries and tables for large datasets

Hue

Web UI to make it easy to work with Hadoop data

Cloudera Impala

Interactive, low-latency SQL queries on HDFS or HBase

Kite SDK

APIs, examples, and docs for building apps on top of Hadoop

Apache Mahout

Scalable machine-learning and data-mining algorithms

Apache Oozie

Workflow scheduler for interdependent Hadoop jobs

Apache Parquet (incubating)

An efficient columnar storage format for nested data

Apache Pig

Data flow language for exploring large datasets

Cloudera Search

Free-text, Google-style search of Hadoop data

Apache Sentry (incubating)

Granular, role-based access control for Hadoop users

Apache Spark

A cluster computing framework for large-scale in-memory data processing in Scala, Java, and Python

Apache Sqoop

Efficient transfer of data between structured data stores (like relational databases) and Hadoop

Apache ZooKeeper

Highly available coordination service for distributed applications

Cloudera also provides *Cloudera Manager* for deploying and operating Hadoop clusters running CDH.

To download CDH and Cloudera Manager, visit <http://www.cloudera.com/downloads>.

REFERENCES

- [1] Go, A., Bhayani, R., & Huang, L. (2009). Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 1-12.
- [2] Tang, H., Tan, S., Cheng, X., A survey on sentiment detection of reviews, Expert Systems with Applications: An International Journal, v.36 n.7, p.10760-10773, September, 2009.
- [3] A. Pak and P. Parouek, "Twitter as a corpus for sentiment analysis and opinion mining," in Proceedings of LREC, vol. 2010.
- [4] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Communications of the ACM, Vol. 51, Iss. 1, pp. 107-113, January 2008.
- [5] S. Ghemawat, H. Gobioff and S-T. Leung, "The Google File System," ACM SIGOPS Operating System Review, Vol. 37, Iss. 5, pp. 29-43, December 2003.
- [6] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in the 26th IEEE Symposium on Mass Storage Systems and Technologies, pp. 1-10, May 2010.
- [7] Bahrainian, S.A., Dengel, A., Sentiment Analysis using Sentiment Features, In the proceedings of WPRSM Workshop and the Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence, Atlanta, USA, 2013.
- [8] "Sentimental Analysis", Inc. [Online]. Available: <http://www.cs.uic.edu/~liub/FBS/sentiment-analysis> [Accessed 23 March 2013].
- [9] (Online Resource) Hive (Available on:<http://hive.apache.org/>).
- [10] (Online Resource)<http://jsonlint.com/>
- [11] (Online Resource)<http://nlp.stanford.edu/software/corenlp.shtml>.

[12] T. White, "The Hadoop Distributed Filesystem," Hadoop: The Definitive Guide, pp. 41-73, Gravenstein Highway North, Sebastopol: O'Reilly Media, Inc., 2010.

[13] (Online Resource) <http://flume.apache.org/>

[14] S. W. Ambler. Relational databases 101: Looking at the whole picture. www.AgileData.org, 2009.

[15] <https://www.tutorialspoint.com/hive/index.htm>

[16] http://www.tutorialspoint.com/apache_flume/

[17] https://www.tutorialspoint.com/apache_pig/

[18] <https://acadgild.com/blog/sentiment-analysis-on-tweets-using-afinn-dictionary/>