DISTRIBUTED FRAMEWORKS TOWARDS BUILDING AN OPEN DATA ARCHITECTURE

Ramu Reddy Venumuddala

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

May 2015

Venumuddala, Ramu Reddy. *Distributed Frameworks towards Building an Open Data Architecture*. Master of Science (Computer Science), May 2015, 47 pp., 6 tables, 33 figures, references, 42 titles.

Data is everywhere. The current Technological advancements in Digital, Social media and the ease at which the availability of different application services to interact with variety of systems are causing to generate tremendous volumes of data. Due to such varied services, Data format is now not restricted to only structure type like text but can generate unstructured content like social media data, videos and images etc. The generated Data is of no use unless been stored and analyzed to derive some Value. Traditional Database systems comes with limitations on the type of data format schema, access rates and storage sizes etc. Hadoop is an Apache open source distributed framework that support storing huge datasets of different formatted data reliably on its file system named Hadoop File System (HDFS) and to process the data stored on HDFS using MapReduce programming model.

This thesis study is about building a Data Architecture using Hadoop and its related open source distributed frameworks to support a Data flow pipeline on a low commodity hardware. The Data flow components are, sourcing data, storage management on HDFS and data access layer. This study also discuss about a use case to utilize the architecture components. Sqoop, a framework to ingest the structured data from database onto Hadoop and Flume is used to ingest the semi-structured Twitter streaming json data on to HDFS for analysis. The data sourced using Sqoop and Flume have been analyzed using Hive for SQL like analytics and at a higher level of data access layer, Hadoop has been compared with an in memory computing system using

Spark. Significant differences in query execution performances have been analyzed when working with Hadoop and Spark frameworks. This integration helps for ingesting huge Volumes of streaming json Variety data to derive better Value based analytics using Hive and Spark.

ACKNOWLEDGMENTS

I would like to take this platform to express my sincere gratitude to my research advisor, Dr. Fu, Song for his continuous and generous effort to support and guide me through out the research study. Without his invaluable assistance and readiness to answer my questions, this would not be possible for me to complete my thesis.

I would like to thank my committee members Dr. Caragea, Cornelia and Dr. Huang, Yan in taking their precious time to review the thesis and chalk out suggestions.

Finally, I would like to thank my parents and friends for their love and constant support through out this journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

"In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a large ox. We shouldn't be trying for bigger computers, but for more systems of computers."

—Grace Hopper

In the current digital world there is a substantial growth in the amount of data that's been generating in the form of structured, unstructured, semistructured data from various sources like sensors, servers, social websites, e-commerce, gaming, and video contents etc. The "Variety, Volume and Velocity" [26] of the Data needs to be stored at a repository, processed and analyzed to derive better Value. The amount of size the disk can store is increasing alarmly in the recent years but, the rate of data transfer has been a concern and is not yet upto the efficient standards [38]. If all the data is stored on a single disk then, performing the read and write access is always a time consuming process. Storing and retrieving the data on a single disk has been a major challenge with respect to cost budget in building the hardware equipment, disk access latency and computational efforts [38]. Distributed System design allows to share the data across the machine nodes and execute the work in parallel [38].

To store the data across multiple disks and accessing them in parallell is always challenging and imposes some challenges. Few problem's of such can be like some storage systems enforce a schema to write the data on to it [38], or there can be a failure in some systems thats been storing the data across the network and it might lead to the loss of a snapshot of the data [38], or how the data can be computed that's been stored across linearly in the network [38]. When working with huge volumes of data it becomes difficult to manage the data across the network and so to move the processing task to the data is pretty cheaper and efficient than moving the data across the network for computation [28, 11, 38]. These concerns have highly motivated in building few open source distributed computing frameworks that can scale linearly across the nodes in distributed fashion on cheap and commodity hardware [38]. Apache^{TM} Hadoop® [11] will address these issues in terms of ”data locality” [28, 38], redundancy by replicating the data blocks across the cluster [28] and provide high data intensive computational analysis [4] to store and retrieve this huge data.

1.1. Thesis Outline

Chapter 2 introduces the MapReduce programming model and Hadoop Distributed File System in detail. Section 2.3 discusses about Hadoop framework which is an open-source implementation of MapReduce and HDFS. The Chapter also talks about few other Hadoop related projects that are used for this thesis study like Sqoop 2.4.1, Flume 2.4.2, Hive 2.4.3, Spark 2.4.4. A clustering framework named Ganglia 2.4.5 is also been discussed at the end of this Chapter.

Chapter 3 talks about the architecture design and enviornment configuration setup that has been used for this thesis study.

Chapter 4 explores an usecase and possible results on the environment thats been setup for this research study.

Chapter 5 concludes the thesis, summarizes the observations and project some future enhancements of the research.

CHAPTER 2

FRAMEWORKS BACKGROUND

This Chapter explains in detail about the distributed frameworks that are used for this research study. Distributed systems can be best understood as a group of machines separated at a different physical locations on network, cooperate with each other to perform a common task or a set of related tasks [21]. As the systems will be distributed across different physical locations some design constraints imposed on these distributed systems are, they need to be reliable, provide high availability of continuous service [11, 28], fault tolerant [39] in recovering from failures and scalable across at ease etc. This Chapter discusses about those distributed programming models and frameworks.

2.1. MapReduce

"MapReduce is a programming model and an associate implementation for processing and generating large data sets" [4]. This model can be scaled across a cluster of cheap and commodity hardware [4, 38]. The model is divided into 2 phases. Map and Reduce phase.

- Map: This phase read the input datachunk and generates a list of intermediate key-value pairs which will then be consumed and processed by the Reducer phase [4]. For example, a text document of urls as an input for Map function will read the data and generate a url pair list. That is, the key is set to the url thats been parsed and the value will be set to 1. For example: If the data is 'hadoopurl, mapreduceurl, hadoopurl', the following pairs will be generated (hadoopurl,1), (mapreduceurl,1), (hadoopurl,1).
- Reduce: This phase will read the intermediate key, list<values> generated by the Map function and aggregates the final result [4]. In this case the input to the Reducer will be (hadoopurl, list<1,1>), (mapreduceurl, list<1>) which will be aggregated to a final result of url count as <hadoopurl,2>,<mapreduceurl,1>.

map	(k1,v1)	list(k2,v2)
reduce	(k2,list(v2))	list(v2)

Table 2.1. MapReduce [4]

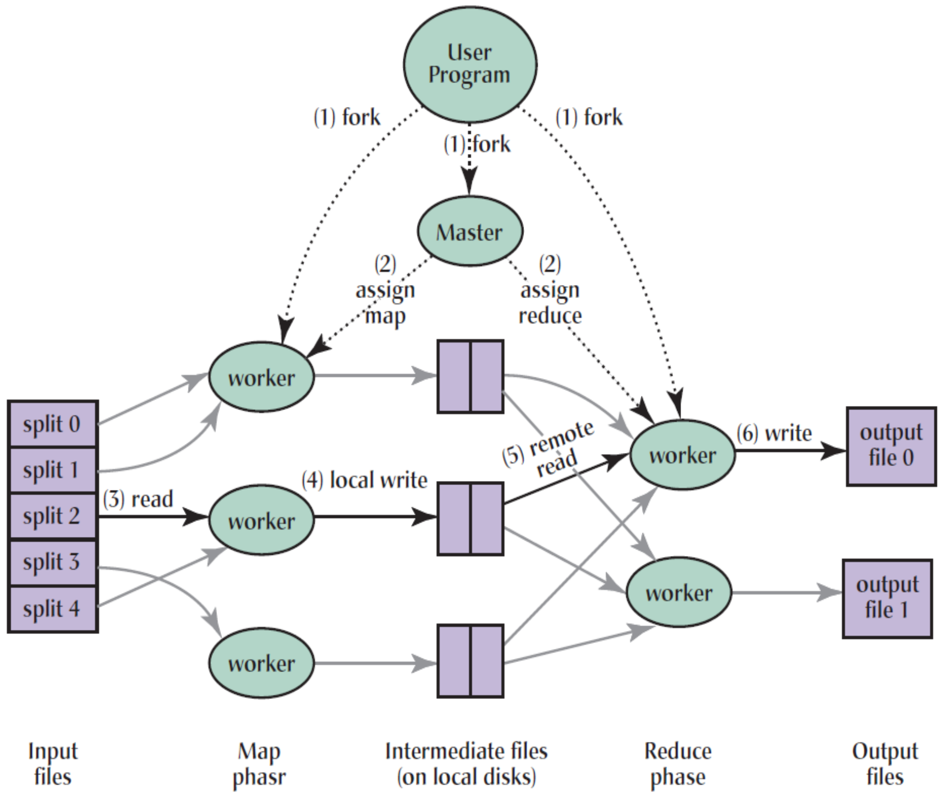


Figure 2.1. Anatomy of a MapReduce Job Execution[4]

2.1.0.1. Anatomy Of MapReduce

The Figure 2.1 describes the Anatomy of a MapReduce job. When a MapReduce job is submitted by the user, a series of events occur during the execution. An overview of the MapReduce is as follows: [4, 22]

(1) The input data for the job to be processed is first split into chunks of a configured size which will then be distributed across the cluster.
(2) Any one of the node is set to act as a Master node and it starts allocating work to the other Slave nodes in the cluster.

(3) The Slave nodes which have been assigned with a map task will now parse the contents of its corresponding input chunk of data and generate an intermediate key/value pairs. This intermediate data will be stored on the Slave nodes.

(4) The buffered intermediate pairs are periodically persisted onto the disk and the Master node is made aware of the location of the data which will then be forwarded to other workers to initate the Reduce task.

(5) The Slave nodes now executing the Reduce tasks will fetch the intermediate key/value pair data by using remote procedure calls, sorts and group the data of the same keys

(6) The reduce task on the Slave node will now run over the list of all the intermediate values of the same reducer, process , aggregates the final result and will flush out to an output file.

(7) When all the map and reduce tasks are succesfull and when there are no more data to process, the user will be notified about the results.

2.2. Hadoop Distributed File System

HDFS™ [28, 11] is a distributed file system component of Apache™ Hadoop® that can source in large volumes of data in the order of Giga to Tera bytes [34] on cheap and low-cost hardware [38]. HDFS has many goals and few of the noteworthy are it is highly “fault-tolerant” [39] and error-prone to failures [11, 8]. It follows “master-slave” architecture [28, 11, 8]. It supports Batch processing rather than interactive use of the data. HDFS follows WORM coherency model for files. WORM is ”write once-read many” [38, 28], that is, file once created and written is not updated any more at record level but can have random read access. It achieves scalability by distributing data across the clusters and reliability by maintaining multiple copies of data [28, 8] in order to avoid failures during computation. HDFS stores file system metadata on a separate machine called “Namenode” (master) [28, 11, 38] and application level data separately on “Datanode” (slave) [28, 11, 38].

2.2.0.2. Architecture

This subsection 2.2.0.2 explains about few components of the HDFS architecture.

- Blocks: The file level data which the disk can accept is, where any application has an access to read and write the data onto it. The smallest unit of data that any disk can accept is referred as its block size. HDFS also deals with the concept of blocks but at a larger magnitude of size and the default value set in Hadoop is 64MB [38, 28, 11] and can be configured to 128MB, 192MB etc. The HDFS block size can be configurable by changing the hdfs-site.xml configuration file in the Hadoop framework. In HDFS, if the file size is less than the configured block size, then it does not occupy the full block size space that has been configured in the configuration file of the underlying storage. That is, if the files are split into 64 or 124 MB blocks and if a file is less than this 64MB/124MB then the whole size will not be used [38].
- NameNode: The "NameNode" [28, 11] works as a master and "manages the file system hierarchy and the mapping of the file blocks to DataNode" [28, 11]. NameNode manages the metadata for all the files and the directories of the file system in the form of 2 files: "fsimage" [28, 38, 11] and "edit log" [38, 28, 11]. The fsimage is a file system image that captures the system namespace and edit log captures the log snapshots when there are any changes to the namespace. There can multiple NameNodes in a cluster and generally its a good practice to maintain multiple NameNodes to support "High Availability" [11] of service in case of failure.
- DataNode: The "DataNode" [28, 11] are the Slave components which store the actual file data. The files on the HDFS are split into one or more blocks and then they land onto the DataNodes. DataNode always sends its metrics and availability in the form of hearbeats to the NameNode. In general there will be only one DataNode per machine in the Hadoop cluster [11].

2.2.0.3. Data Flow

The Figure 2.2 describes the HDFS data flow architecture. If a user or client wants to access and create a file in the HDFS, the files are first split into blocks and for each block of the file to be stored on the DataNode, the NameNode will look at the current namspace data

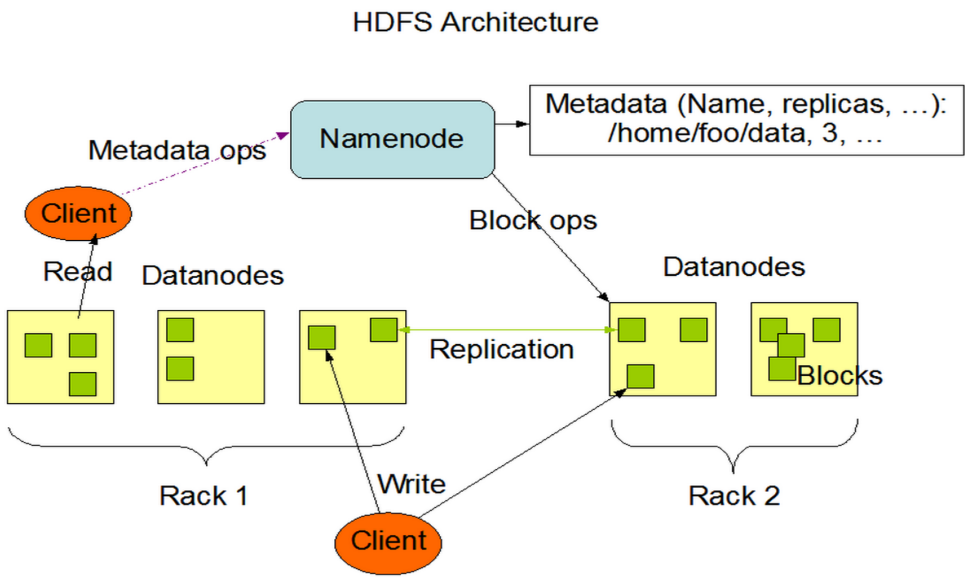


FIGURE 2.2. Hadoop Distributed File System Data Flow[12]

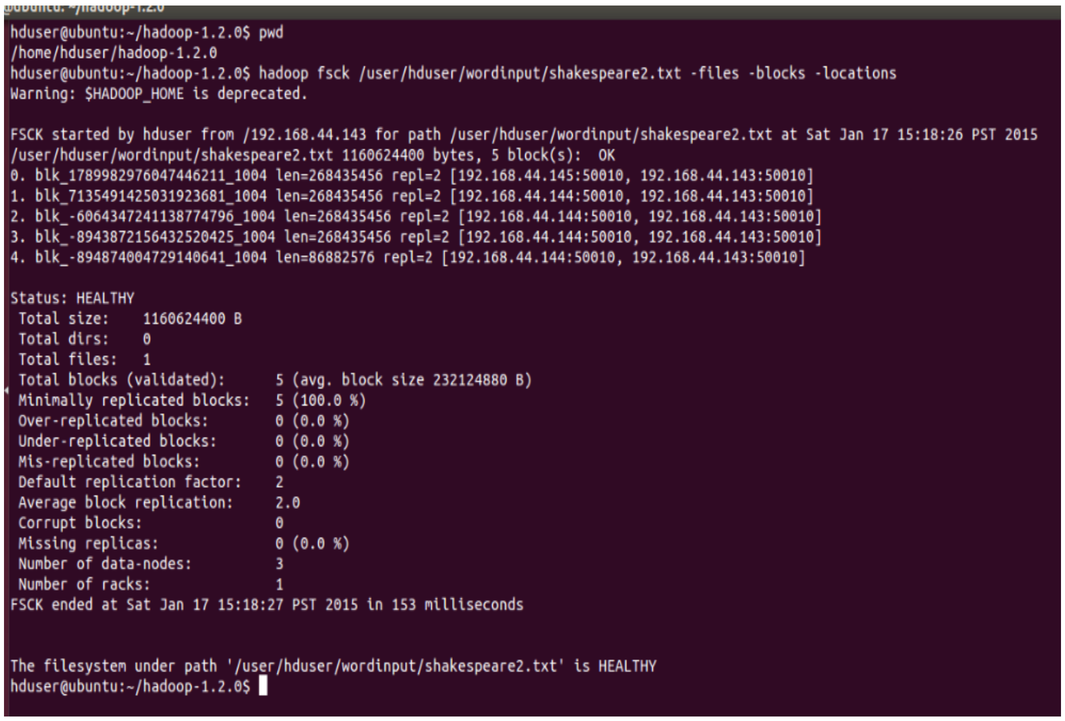
for the suitable available DataNodes. The client will get this block report from NameNode and then the blocks are buffered through and distributed onto the available DataNodes. The default replication specified by HDFS is 3. [11].

HDFS's "fsck" (file system check) [11, 38] command utility will specify the files, blocks and locations on HDFS. The Figure 2.3 shows the files, block locations metrics.

```
$ hadoop fsck <hdfs directory> -files -blocks -locations
```

2.3. Introduction To Hadoop

The Apache™ Hadoop® [11] is an open source framework mainly intended to store huge amount of data reliably using "Hadoop Distributed File System" (HDFS) [28, 11] and process the data across the computing clusters using "MapReduce" (MR) [4] programming model. Hadoop was originally started by Doug Cutting as a basis for Apache Nutch™ a web search engine project [20, 38]. Web is full of pages and the team at Nutch has realized the difficulty to search and index those huge amount of web pages efficiently. In 2003, the team at Google has published a paper called "Google File System" (GFS) [8] which is a distributed file system at Google and Nutch has implemented the idea to its own project which turned into "Nutch Distributed File System" (NDFS) [20, 38]. In 2004, Google published a paper



```
hduser@ubuntu:~/hadoop-1.2.0$ pwd
/home/hduser/hadoop-1.2.0
hduser@ubuntu:~/hadoop-1.2.0$ hadoop fsck /user/hduser/wordinput/shakespeare2.txt -files -blocks -locations
Warning: $HADOOP_HOME is deprecated.

FSCK started by hduser from /192.168.44.143 for path /user/hduser/wordinput/shakespeare2.txt at Sat Jan 17 15:18:26 PST 2015
/user/hduser/wordinput/shakespeare2.txt 1160624400 bytes, 5 block(s):  OK
0. blk_1789982976047446211_1004 len=268435456 repl=2 [192.168.44.145:50010, 192.168.44.143:50010]
1. blk_7135491425031923681_1004 len=268435456 repl=2 [192.168.44.144:50010, 192.168.44.143:50010]
2. blk_-6064347241138774796_1004 len=268435456 repl=2 [192.168.44.144:50010, 192.168.44.143:50010]
3. blk_-8943872156432520425_1004 len=268435456 repl=2 [192.168.44.144:50010, 192.168.44.143:50010]
4. blk_-894874004729140641_1004 len=86882576 repl=2 [192.168.44.144:50010, 192.168.44.143:50010]

Status: HEALTHY
 Total size:    1160624400 B
 Total dirs:    0
 Total files:   1
 Total blocks (validated):      5 (avg. block size 232124880 B)
 Minimally replicated blocks:   5 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    2
 Average block replication:     2.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          3
 Number of racks:               1
FSCK ended at Sat Jan 17 15:18:27 PST 2015 in 153 milliseconds


The filesystem under path '/user/hduser/wordinput/shakespeare2.txt' is HEALTHY
hduser@ubuntu:~/hadoop-1.2.0$
```

Figure 2.3. Hadoop File System Check Utility [11, 38]

[4] and introduced the term "MapReduce" (MR) [4]. By early 2005, Nutch developers has implemented the MapReduce programming model and integrated their algorithms to run on NDFS and MR. In February 2006, NDFS and MR has moved out of Nutch and formed an independent project named Apache Lucene^{TM} [38].

In January 2008, Hadoop has become a pivotal project at Apache and NDFS was renamed to HDFS or Hadoop Distributed File System. The framework's design motivation is to scale linearly from a single node to thousands of nodes [34] to support both storage (HDFS) and computation (MR) efficiently. The framework is designed to detect and handle node failures [4, 28] and to provide high-availability of service [11, 36, 38]. Many companies like Yahoo, Facebook, Linkedin etc have started using Hadoop and eventually it gained a huge popularity. It's open source, free and can work on cheap commodity hardware as well. This section 2.3 is heavily borrowed from [38] (9-10).

2.4. Hadoop-related Projects

There are few distributed frameworks under Apache^{TM} that are related to Hadoop in terms of storage and compute point of view. They can be termed as ”Ecosystems” [11] related to Hadoop under Apache. Some of them under this research work has been discussed below.

2.4.1. Sqoop

From many years till date, large scale Companies store their transactional and operational data on traditional Database systems like Oracle, Greenplum and Postgres etc. Since, Hadoop support storage in the form of HDFS and can allow many different file formats like text, json, avro and binary, there has been an integration that's been developed by the database vendors to work with Hadoop systems. Hadoop can manage and process the data from the Databases . Apache Sqoop^{TM} “SQL to Hadoop,” [33] is a tool intended to bring in data from Database onto Hadoop and vice-versa [33, 31, 38]. The entire framwork of Sqoop is written using Java. The main execution framework behind the scenes of Sqoop import and export jobs is MapReduce and it support fault-tolerance and reliability during the transfer process [31]. Sqoop extends its support to transfer of data from Database to Apache Hive^{TM} and Apache HBase^{TM} as well, which are the ecosystems that sit on top of the HDFS storage.

2.4.1.1. Sqoop Import Architecture

Sqoop import job takes few input arguments and transfer the row level table data from the Database on to Hadoop (HDFS, Hive, HBase) by running MapReduce jobs. Few input parameters for the Sqoop import job are –connect (jdbc-uri), –username (db username), -P (password), –table (table data to retrieve), –driver (jdbc driver), –target-dir (target directory on Hadoop) [33, 31].The Database specific JDBC drivers need to be downloaded and installed in the Sqoop “lib” folder of the Client before running the import or export jobs. Based on the –connect string url that's been specified in the input parameter, the corresponding driver gets loaded before the start of the import job [38].

The graphical representation of the import job is shown in Figure 2.4. Sqoop import

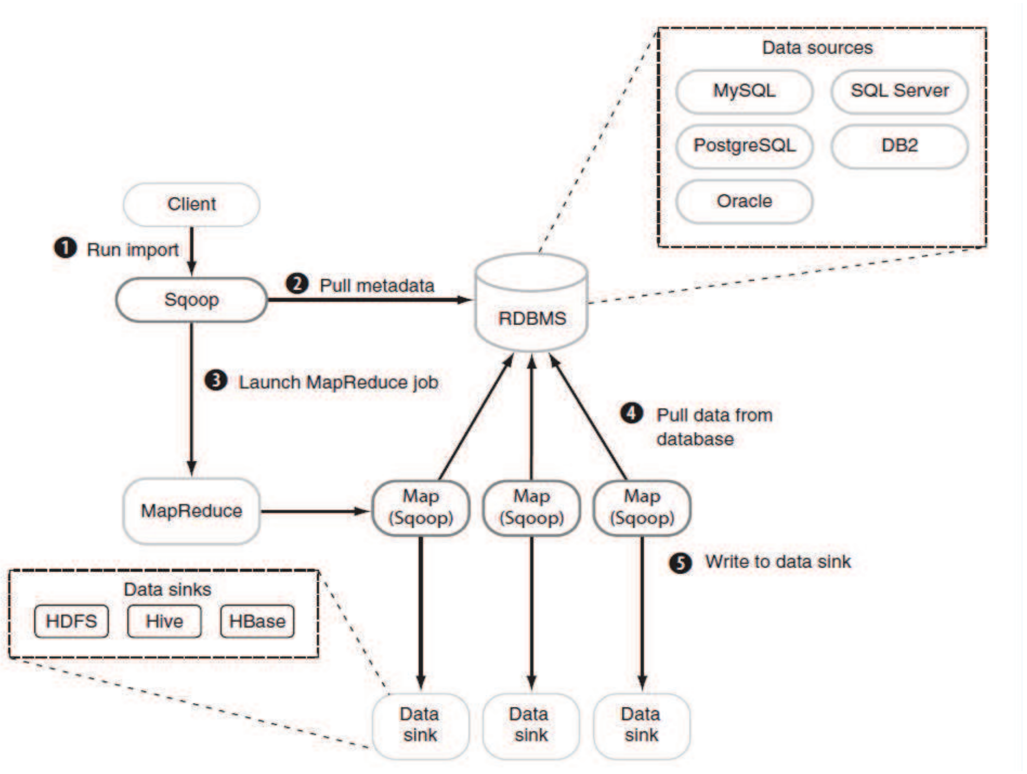


Figure 2.4. Sqoop Import Process [16]

job can be broken down in to stages before the actual import can happen. Firstly, it uses JDBC to fetch the table schema that it has to import from the database [38]. The fetched SQL data types are then mapped to hold the Java data types for MapReduce applications functionality. Secondly, Sqoop will generate a record container class [38] to interpret and hold the table fields during the import. Finally, Sqoop import launches the MapReduce job to read the table data using JDBC and source it on to Hadoop. The Sqoop import jobs can also be controlled with few other input parameters like number of mappers to run (-n) during the MapRedue job, "–direct" mode for faster data transfer [33, 31].

2.4.1.2. Sqoop Export Architecture

The export process diagram is shown in the Figure 2.5. Sqoop export job takes few output arguments and transfer the data from Hadoop to Database Systems. The export job requires the –export-dir argument to specify the HDFS directory location to be exported on to target Database System. Sqoop export job is broken down in to stages before export can

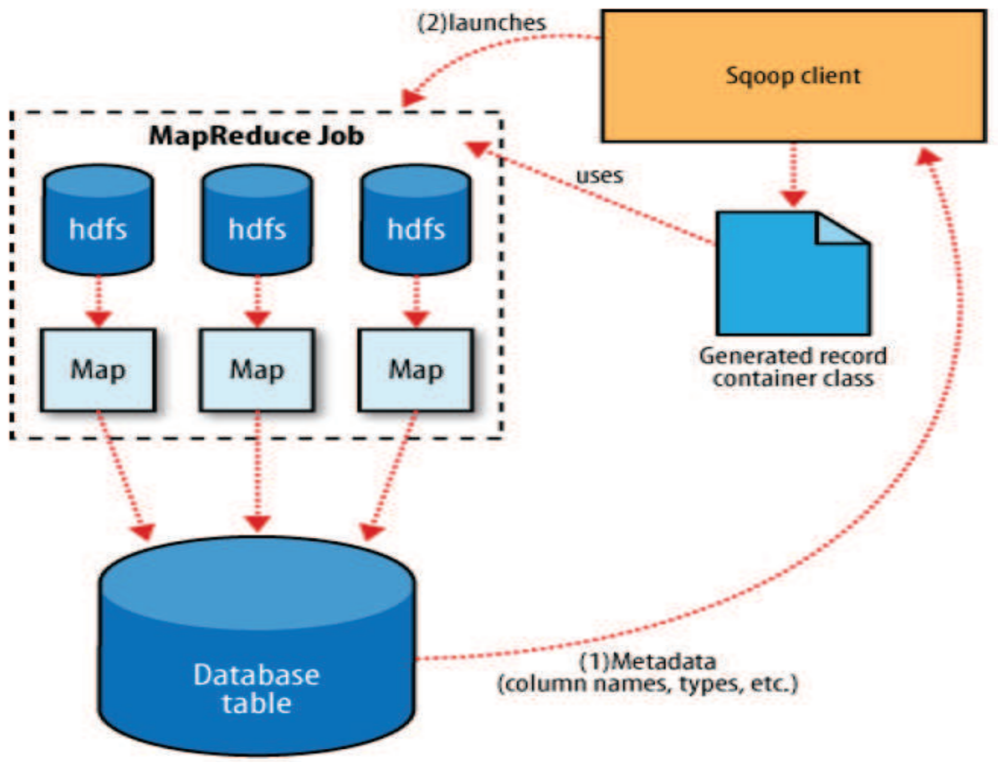


FIGURE 2.5. Sqoop Export Process [16]

happen and follows similar nature as that of an import job. Based on the specified –connect argument url, Sqoop picks up the JDBC strategy and loads the related drivers. Based on the target table schema, it generates a Java container class to parse the records from HDFS files and load them to the mapped data types into the target table. Then, finally MapReduce jobs are fired to read the data from the files, parse them based on the container class and executes the export job to write data onto Database [38].

2.4.2. Flume

Apache Flume^{TM} is a reliable and distributed data ingestion tool that can ingest streaming and aggregate large amounts of data from different sources onto target data store" [6, 17]. There are variety of data sources generating data in the form of server logs, user content on social media platform, user engagement on web service applications, network systems data. All these systems can generate data in the form of structured text to unstructured format. There can be Custom sources which can be configured in Flume to generate the data as well as to source in onto data store. Flume has the ability to scale horizontally to handle large data volumes and can gurantee data delivery during the ingestion [17]. The below section explains about the Flume Data Achitecture in detail.

2.4.2.1. Flume Architecture

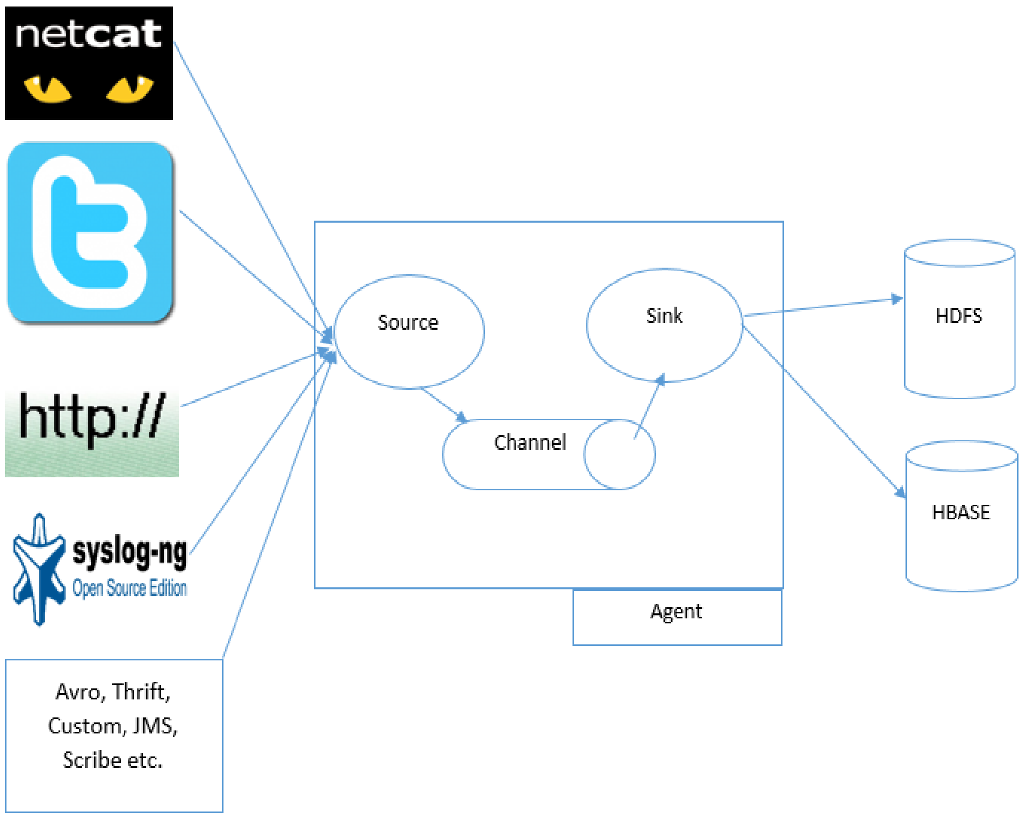


FIGURE 2.6. Flume Architecture [16, 6]

The main components of Flume are Event, Source, Channel, Sink and Agent [14, 6, 17].

- Event: The topics delivered by the Flume are called Events. Event is a smallest unit of data occurence.
- Source: The events generated by the external sources like Web server, Twitter, syslog and netcat etc are taken by the Flume Source which are then stored onto one or more Channels in the next data flow.

- Channel: A Channel is a medium to store the event before it's being consumed by the target Sink. Different type of channels exists in Flume, some of them are Memory Channel, File Channel and JDBC channel etc.
- Sink: The Sink finally aggregate the event from the Channel and dumps it into an external repository like HDFS, HBase or forwards it to the Flume Source of the next Agent in the flow.
- Agent: An Agent is Java Virtual Machine process running in flume that integrates the Source, Sink, Channel components together as a unit through which events flow from external sources to the next hop of the data flow [17, 6]

Flume achieves reliablity by ensuring that the events are removed from the previous agent staged channel only if they are transferred to the channel of the next available agent or onto a target repository. Since, Flume stages the event in a channel, there can be a case of failure and loss of events. Flume ensures recoverability by supporting a durable File Channel which provides checkpointing on a local file system for recovery. Flume also provides Memory Channel but, it has a limitation on the queue capacity at a cost of high throughput and no gurantee in recoverability. That is, if the Agent experiences any failure then the events in the channel are lost and cannot be recovered. This information is borrowed from [6]

Flume Agent can be created by creating a new configuration file or by editing the existing configuration file stored in the "conf" directory of Flume installation. The configuration file should include the properties of Source, Channel and Sink. This research study has used Source as a Twitter Source provided by Cloudera [3], Channel as a File Channel [6] for recoverability and Sink as HDFS for storage [6]. The events can be generated by triggering the Flume agent and Flume ships with a shell script called "flume-ng" which is located in the "bin" folder of the Flume installation. The arguments to be specified for the schell script are -c (conf folder location), -f (configuration file), -n (agent name). The detailed configuration of the Twitter Agent and execution is discussed in the Environment setup section section 3.2.

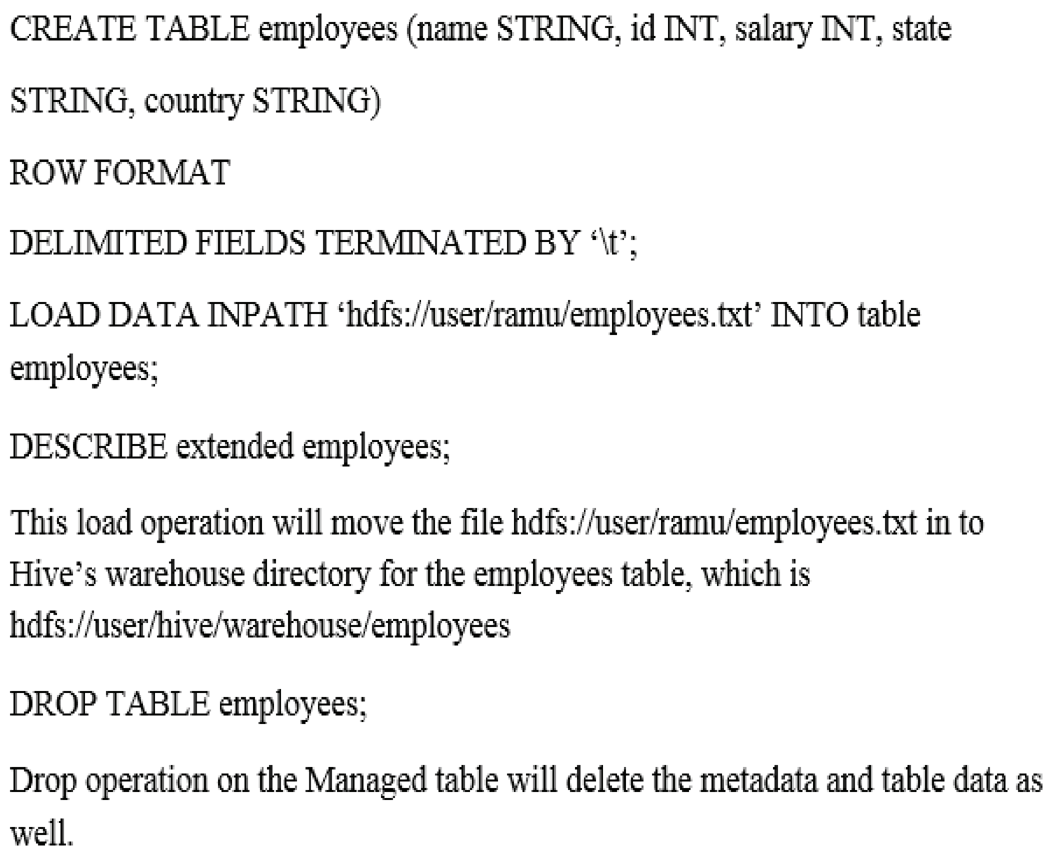
2.4.3. Hive

Apache Hive^{TM} is a framework for data warehouse processing that sits on top of Hadoop. Hive was initially developed by the team at Facebook to analyze the huge volumes of data that they stored on HDFS. Hive provides a SQL like interface called HiveQL [32] to run the queries. These HiveQL queries generates MapReduce jobs that run on the Hadoop cluster. Hive is now popular in many organizations because of it data processing ability and is now a top level project at Apache^{TM} [38] .

2.4.3.1. Data Model

The Data on HDFS is used as a reference on Hive to organize it into Tables, Partitions and Buckets [38, 32].

- Table: "Tables in Hive are analogous to the tables in relational databases" [32]. The Tables in Hive can be constructed depending on the data thats been residing at a physical location and can be either from HDFS, local file system, S3 (Amazon storage systems) or on any other Hadoop related filesystems. The data format is useful in determining the metadata and helps in constructing the Hive Tables efficiently by using the Hive data types. The Hive Table metadata generated from the file systems will be stored in a Database but not on the HDFS. Hive metastore to store this metadata is categorized as embedded, local and remote metastores [38]. Hive support embedded metastore as a default metastore by using Derby Database which is just local to the single level user. Usually, in bigger Production environments there will be multi users accessing the Hive service, so the "local metastore" [38] will be configured to a separate Database Servers like MySQL etc for better management. Hive support both Managed Tables and External Tables. The default Table of Hive is a Managed Table and when file system data is loaded into a Managed table, Hive moves the file data in to its warehouse directory [38, 1]. Hive moves all the files that are used to create tables from it are stored in its default warehouse directory location "hdfs://user/hive/warehouse/" [38, 1] and can also be configured to some other location using “hive.metastore.warehouse.dir” [38, 1] property in hive−site.xml file of Hive installation. Some DDL and DML on Managed table is shown in the Figure 2.7.



```
CREATE TABLE employees (name STRING, id INT, salary INT, state
STRING, country STRING)
ROW FORMAT
DELIMITED FIELDS TERMINATED BY ‘\t’;
LOAD DATA INPATH ‘hdfs://user/ramu/employees.txt’ INTO table
employees;
DESCRIBE extended employees;
```

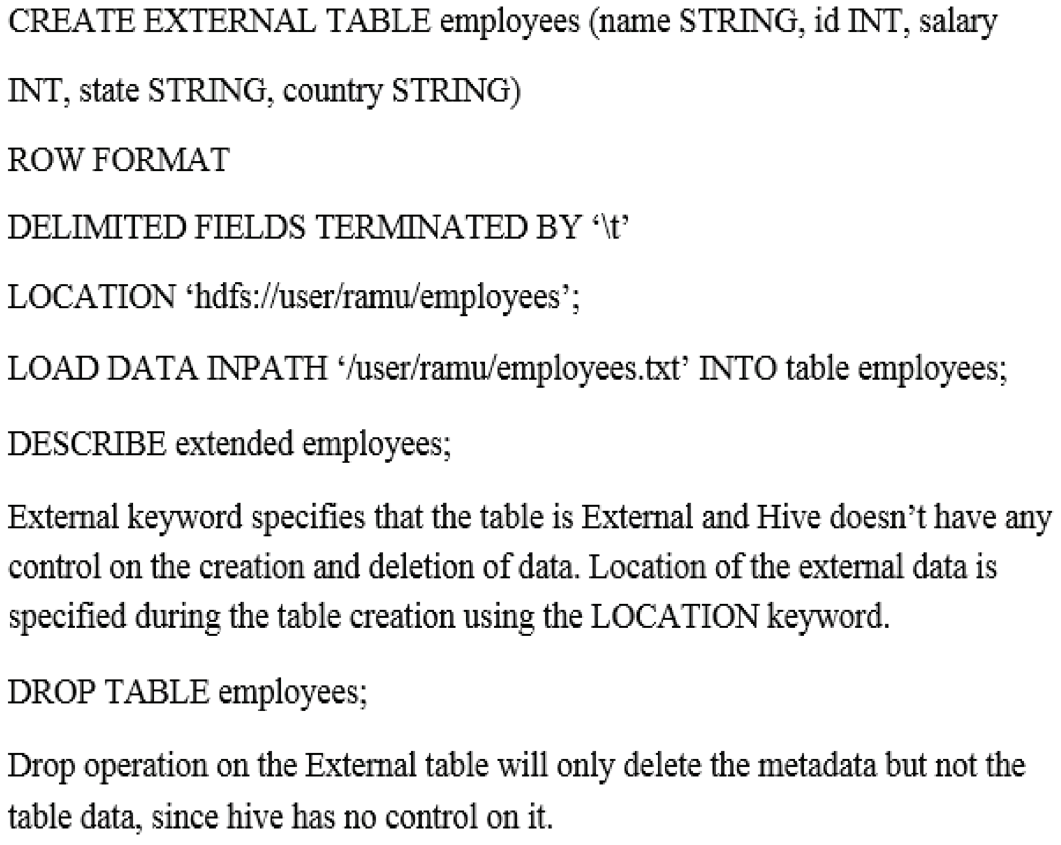
This load operation will move the file hdfs://user/ramu/employees.txt in to Hive’s warehouse directory for the employees table, which is hdfs://user/hive/warehouse/employees

```
DROP TABLE employees;
```

Drop operation on the Managed table will delete the metadata and table data as well.

Figure 2.7. Hive Managed Table [38, 5, 1]

In Figure 2.7 the CREATE statement is used to generate the table named employees with the columns as name, id, salary, state and country. The data can be loaded onto this Hive table using the LOAD statement. Data to be loaded can reside in the local file system or on HDFS. If the data is loaded from a local file system then, “LOAD DATA LOCAL INPATH” [32, 1, 38] syntax is used. If the data is already on HDFS then “LOAD DATA INPATH” [32, 38, 1] is used. So, when you LOAD data onto Managed table, the file ‘hdfs://user/ramu/employees.txt’ is moved into Hive warehouse directory ‘hdfs://user/hive/warehouse/employees’. The table details like locations, size of the data can be DESCRIBE statement explains about the table details in a neat formatted way. "DROP" [1, 38] statement on a Managed table will drop the metadata and table data as well.



```
CREATE EXTERNAL TABLE employees (name STRING, id INT, salary
INT, state STRING, country STRING)
ROW FORMAT
DELIMITED FIELDS TERMINATED BY '\t'
LOCATION 'hdfs://user/ramu/employees';
LOAD DATA INPATH '/user/ramu/employees.txt' INTO table employees;
DESCRIBE extended employees;
```

External keyword specifies that the table is External and Hive doesn't have any control on the creation and deletion of data. Location of the external data is specified during the table creation using the LOCATION keyword.

```
DROP TABLE employees;
```

Drop operation on the External table will only delete the metadata but not the table data, since hive has no control on it.

Figure 2.8. Hive External Table [1, 5, 38]

The Figure 2.8 describes an External table. External tables are those where Hive doesnot manage the Tables, but the user will have the complete control on the creation and deletion of the data. During the creation of an External table, "LOCATION" [1, 5, 38] keyword has to be explicitly specified to point to the data location. If the application depends only on the Hive to manage the data then its suggestable to depend on Managed tables. If other data processing tools and applications are also using the same dataset for analysis, then its a good practice to create External tables because, if the External tables are dropped table metadata is lost and the data still persists in that location.

- Partitions: Partitioning is a technique on Hive which gives a table for better managability to organize it and table data is sub divided into modular parts in the form of files depending on the value of a Partition column [38, 32]. Partition columns can be on date, id etc. When the table is Partitioned into files, the query is interested in looking at that part of the data stored in the specific Partition location and helps in querying the data faster..

- Buckets:

2.4.3.2. SerDe

SerDe refers to "Serializer/Deserializer" [10, 32] which helps Hive to to interpret the table during the read and write operations [32, 10]. Hive currently supports TextInputFormat and TextOutputFormat as the default format to read and write HDFS files [10]. If the data to be loaded from HDFS is in another format, Hive has to interpret the data format, so SerDes are used based on the format of the data. Hive currently supports SerDes for few format like Avro, ORC, Parquet and CSV etc [5]. A custom SerDe can also be created in Hive [5].

2.4.4. Spark

Apache Spark^{TM} is a general-purpose and fast in memory cluster computing platform offering simple api in Java, Scala, Python and SQL [15]. Spark backs up its intensive computations by extending the MapRedce programming model [15, 42]. When it comes to speed, Spark offers the ability to run the massive stream, iterative type of computations in memory. The Spark system execution is faster than aplication workloads when running with MapReduce on disk [15, 42]. Spark is designed to support high workload jobs like streaming, batch and iterative in a same engine called "Spark Engine" [15] and reduces the management burden of maintaining separate tools 2.9 [15]. Spark can also run in Hadoop clusters and can access any Hadoop related file systems [15].

- Spark Core: "Spark Core" [15] is the heart of Spark and is a single functinal unit that takes care of all the scheduling of tasks, memory management, recovering from failure and working with the storage systems [15]. The basic building API

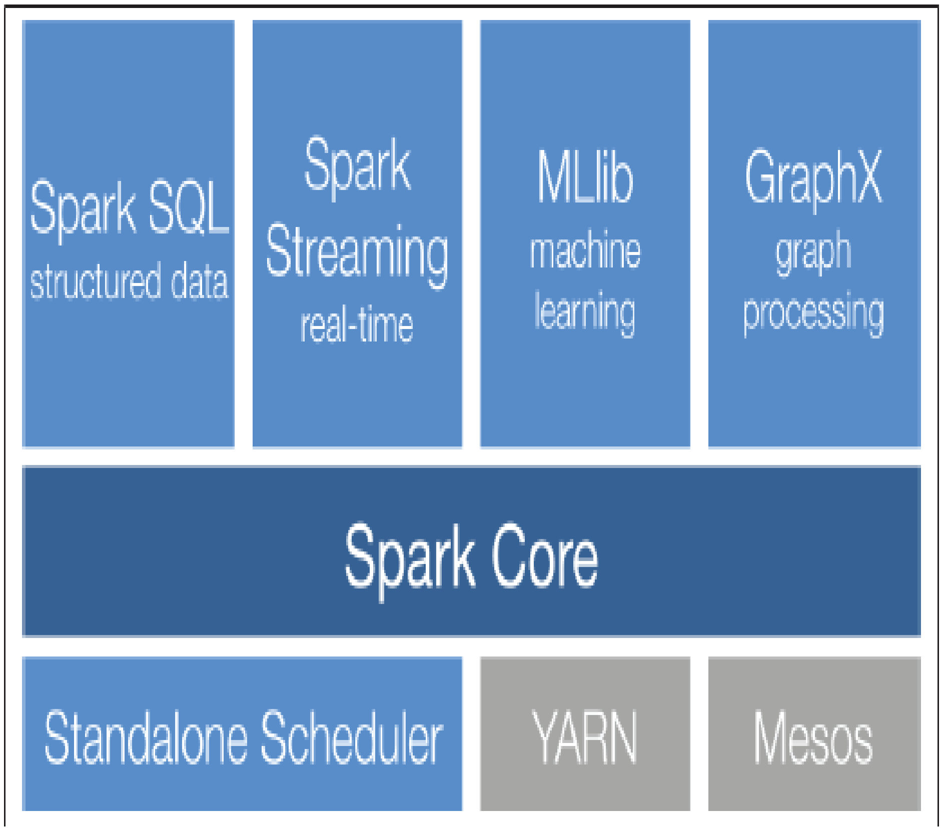


FIGURE 2.9. Spark Stack [15]

abstraction for the Spark Core are "Resilient Distributed Datasets" (RDDs) [41, 15]. These RDDs are fault tolerant, distributed across the cluster and can be computed in parallel [15, 41].

- Spark SQL: "Spark SQL" [15, 30] is a Sql variant of Spark distribution . It allows Sql type query interface to interact with the data and also it extends Hive variant of Sql (Hive QL). Spark SQL extends and provides the ease to access different kinds of available data sources like Hive tables, Parquet and JSON [15, 30].
- Spark Streaming: Streaming data provide valuable information for analysis. "Spark Streaming" [15, 29] is a component of Spark which allows to process the Streaming data. Data Streams can be anything in the form of web logs from the System Servers, or live feed twitter data or can be user updates of a web application. Spark Streaming API allows to manipulate such Data streams and extends the Spark core

RDD API [15]

- MLlib: Spark ships with a machine learning package called ”MLlib” [15, 29] which includes few machine learning applicationfeatures like classification , regression, clustering and collaborative filtering. All these machine learning techniques can be distributed efficiently in linear fashion onto a cluster of machines [15].
- GraphX: Spark provides graph processing by using ”GraphX” [15, 29] library. GraphX library uses the base Spark RDD abstraction API . Some of the graph algorithms provided by graphX are PageRank and traingle counting etc [15].

2.4.5. Ganglia

When working with large cluster of distributed systems or in any data center, there can be a high chance of systems failure due to several reasons like power failure etc. In order to provide a better Service of an application system, its always a good practice to keep track of the computing clusters or grids for better decision making in the event of any failure. There are several distributed monitoring systems that helps to keep track of the metrics and Ganglia is one of them [27].

The machines in the cluster send and receive system level metrics like, cpu, memory and network etc. There are few components in Ganglia that need to be configured to monitor these metrics. They are gmetad, gmond and a web interface called ganglia-web [27].

- gmond: gmond is a Ganglia monitoring daemon and this service has to be installed on all the machines in the cluster in order to monitor their metrics during system work loads.
- gmetad: gmetad is a Ganglia meta daemon which consumes all the metrics thats been generated by the gmond daemons in the cluster. This can be analagous to a master node of Ganglia system. Gmetad uses RRD tool which is a “Round Robin Database” [9, 27] to store all the consumed metrics from the gmond daemons.
- ganglia-web: ganglia-web has to access the RRD data [9, 27] to show the consolidated metrics on a web interface. So, this deamon sits on the same system where gmetad is configured earlier and provides a continuous web visualization of the

metrics data.

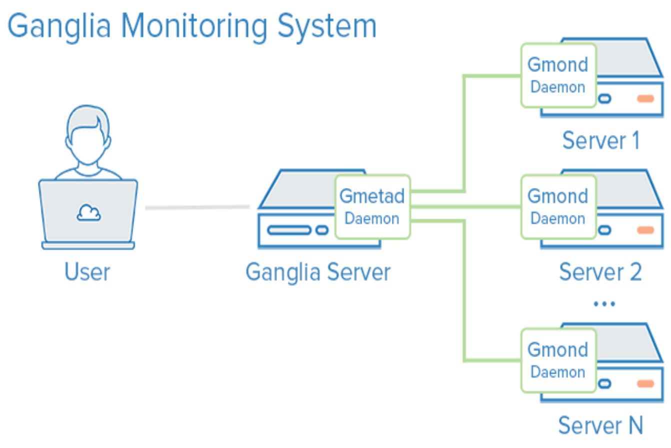
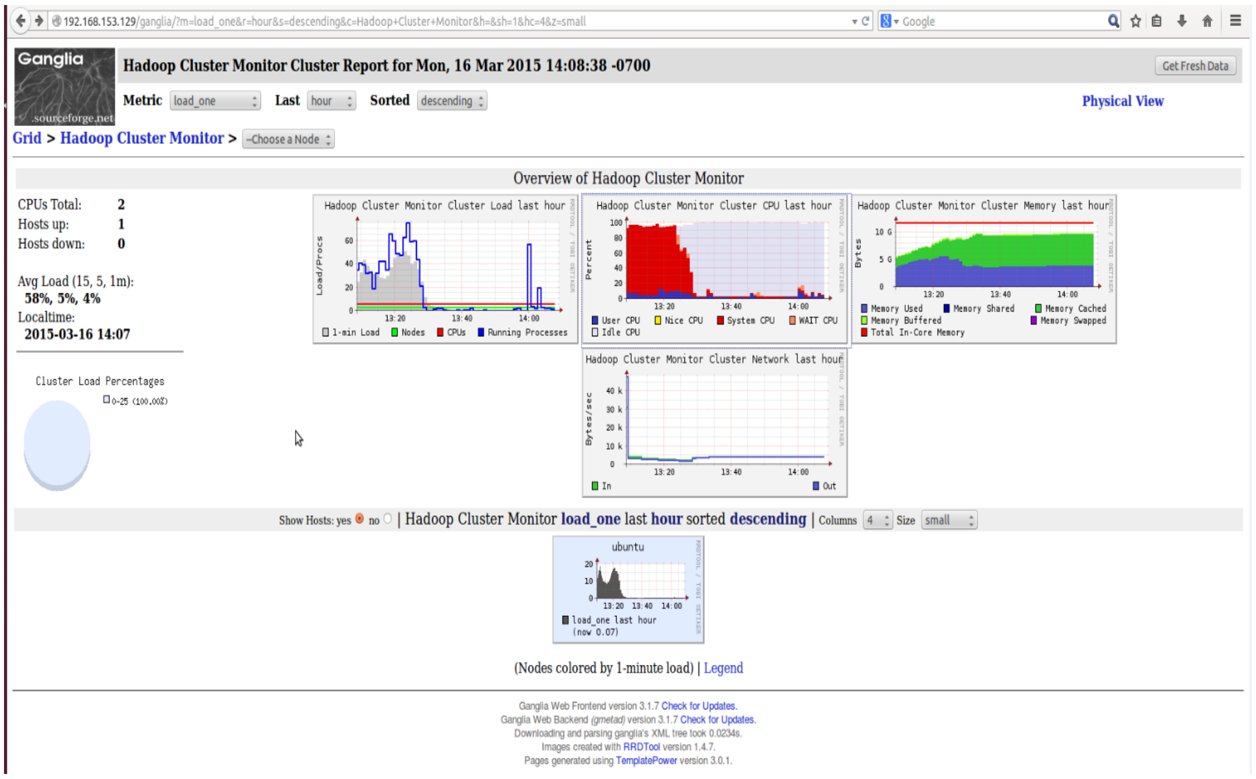


FIGURE 2.10. Ganglia Monitoring [27]


192.168.153.129/ganglia/?m=load_one&r=hour&s=descending&c=Hadoop+Cluster+Monitor&h=&sh=1&hc=4&z=small

Hadoop Cluster Monitor Cluster Report for Mon, 16 Mar 2015 14:08:38 -0700

Metric load_one **Last** hour **Sorted** descending

Grid > Hadoop Cluster Monitor > --Choose a Node

Overview of Hadoop Cluster Monitor

CPUs Total:	2
Hosts up:	1
Hosts down:	0

Avg Load (15, 5, 1m):
58%, 5%, 4%
Localtime:
2015-03-16 14:07

Show Hosts: yes [✓] no | Hadoop Cluster Monitor **load_one** last **hour** sorted **descending** | Columns 4 Size small

(Nodes colored by 1-minute load) | Legend

Ganglia Web Frontend version 3.1.7 Check for Updates.
Ganglia Web Backend *(gmetad)* version 3.1.7 Check for Updates.
Downloading and parsing ganglia's XML tree took 0.0234s.
Images created with RRDTool version 1.4.7.
Pages generated using TemplatePower version 3.0.1.

FIGURE 2.11. Ganglia Visualization of Hadoop Cluster

The Figure 2.11 is a Ganglia web home interface of Hadoop cluster thats been configured for this research study. The url to the web interface is “http://gmetad−ip/ganglia”. It shows network load, process load, cpu load, memory load and few others can be configured in the interface. The daemons in Hadoop like NameNode, JobTracker, TaskTracker etc can all be monitored in Ganglia by modifying “hadoop-metrics.properties” configuration file in the conf folder of Hadoop installation.

CHAPTER 3

ARCHITECTURE DESIGN AND ENVIRONMENT SETUP

3.1. Architecture Design

In this section the detailed architecture design of the research is explained. The main aim of this Data Architecture design is to scale the available exisiting open source distributed technology frameworks and process them efficiently on a low commodity hardware. The key architecture components are divided into Ingesting Data, Managing Data and Accessing Data. Several components are integrated to achieve these features at a single level of design on virtualization environment. This research deals with both the structured text and semi-structured streaming json data. Hadoop Distributed File System (HDFS) is the storage for most part of this research. Using this Data Architecture a use case has been developed for analysis which is discussed in Chapter 4

- Sourcing Data: The Data Ingestion component of this research study is mainly focussed on bringing the data from a Relational Database and from Twitter streaming API. Sqoop is a framework which enables to bring the data from any Relational Database onto Hadoop. Flume is another technique which can ingest streaming data onto different target stores. Hadoop file system commands are used to move data from local file system onto HDFS. These frameworks have been configured to bring the Data onto Hadoop.

Framework	Functionality
Sqoop	To ingest text data from Database
Flume	To bring Twitter streaming data

Table 3.1. Ingestion Component.

- Storage Management: The data ingested from the Database and Twitter Streaming is stored onto HDFS. The reason to store on HDFS is because of its reliability

[28, 11], can support different formats of data and can provide replication of the data. The Streaming data from Twitter is in the form of Json and its hard to interpret the schema of Json data. Few Databases, to an extent provide collection data types to support json format data, but its really challenging to create table schemas for them. HDFS allows to store structured and unstructured data, which later can be managed by using the ecosystems like Hive, HBase based on the requirement.

- Data Access: In this research, Hive has been extensively used to manage both the text and json data thats been ingested using Sqoop and Flume onto HDFS. Hive provides a SQL like interface to access the data. Hive supports collection data types like Array, Map and Struct which help in creating table schemas easily when dealing with semi structured kind of Json data. At an higher level, to differentiate the query execution times, Spark is also been considered in this research. The detailed Architecture is shown below in Figure 3.1 [24].
- Cluster Monitoring: In this research, Apache Hadoop 1.2.0 version is configured on a 3 node VMware cluster. Monitoring the clusters is a good norm at an enterprise level to constantly check and take appropriate actions during extra loads on the machines for efficient functioning and reducing the down time of machines due to failures. This research study has considered Ganglia, "a distributed monitoring system for high-performance systems like clusters and Grids" [7].

Several Hadoop Distributions are availble in the Market that offer these component features in their platform at an enterprise scale. Few of them are Cloudera, Hortonworks, MapR, Pivotal. This architecture study is taken reference using the Data Architecture of Pivotal Data Lake [24]. The main intention of this research is to use these distributed frameworks on low commodity hardware that can scale from a single laptop to cluster of physical machines or onto Virtualization environments.

3.2. Environment Setup

This research in its entirety is done on VMware 11 [37] virtual machine instances. A 3 node cluster with Ubuntu 12.04 (Precise Pangolin) Operating System is been setup to

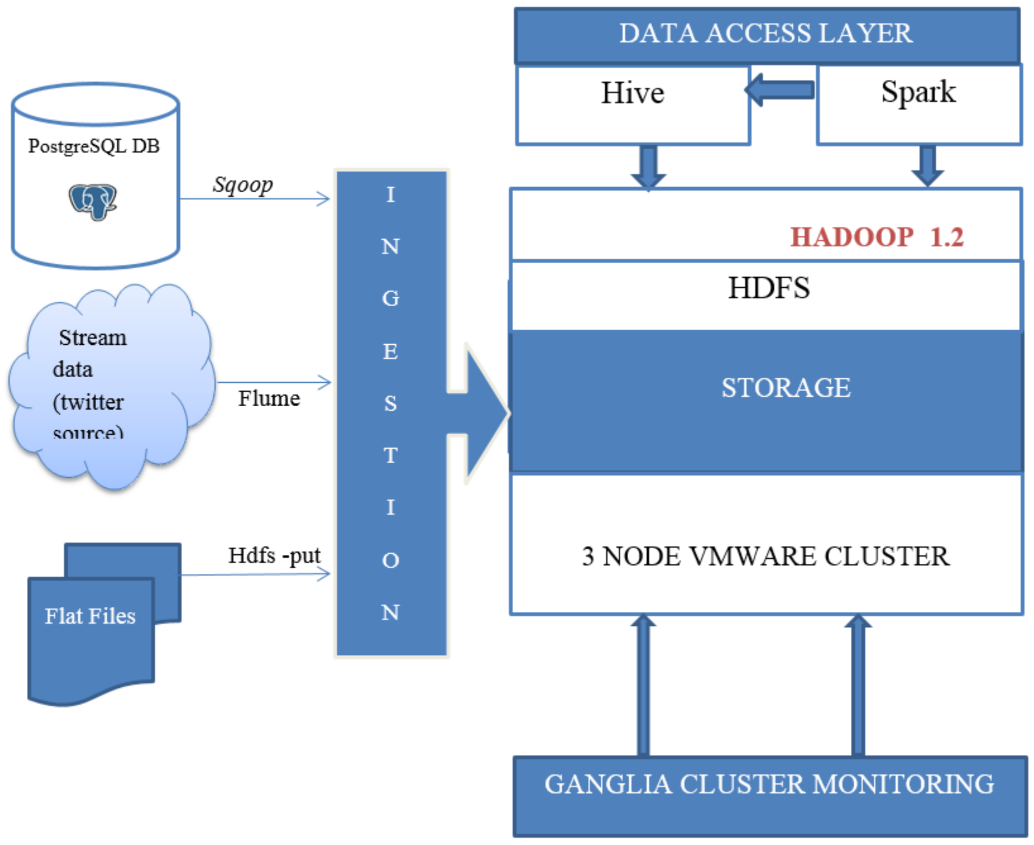


FIGURE 3.1. Data Architecture

configure all the open source frameworks. The Ubuntu instances are configured with Static IP addresses to have better maintainability of the cluster, disk capacity for storage, memory for processing and processors. The Hardware details of the instances are mentioned in the below Table 4.1

Instance Name	Hard Disk Capacity	Memory	Processors	IP
master11	25GB	6GB	2	192.168.153.129
slave11	20GB	3GB	2	192.168.153.130
slave22	20GB	3GB	2	192.168.153.131

TABLE 3.2. Hardware Configuration.

The Table 3.3 depicts the components with respective versions thats been installed and configured for this research study.

Component	Version
OS	Ubuntu 12.04
Java	Openjdk-7-amd64
Apache Hadoop	1.2.0
Apache Sqoop	1.4
Apache Flume	1.4
Apache Hive	0.13
Apache Spark	1.2
Apache Oozie	3.3
Ganglia	3.4
Apache Maven	3.2.1

TABLE 3.3. Cluster Components.

3.2.1. Apache Hadoop 1.2.0

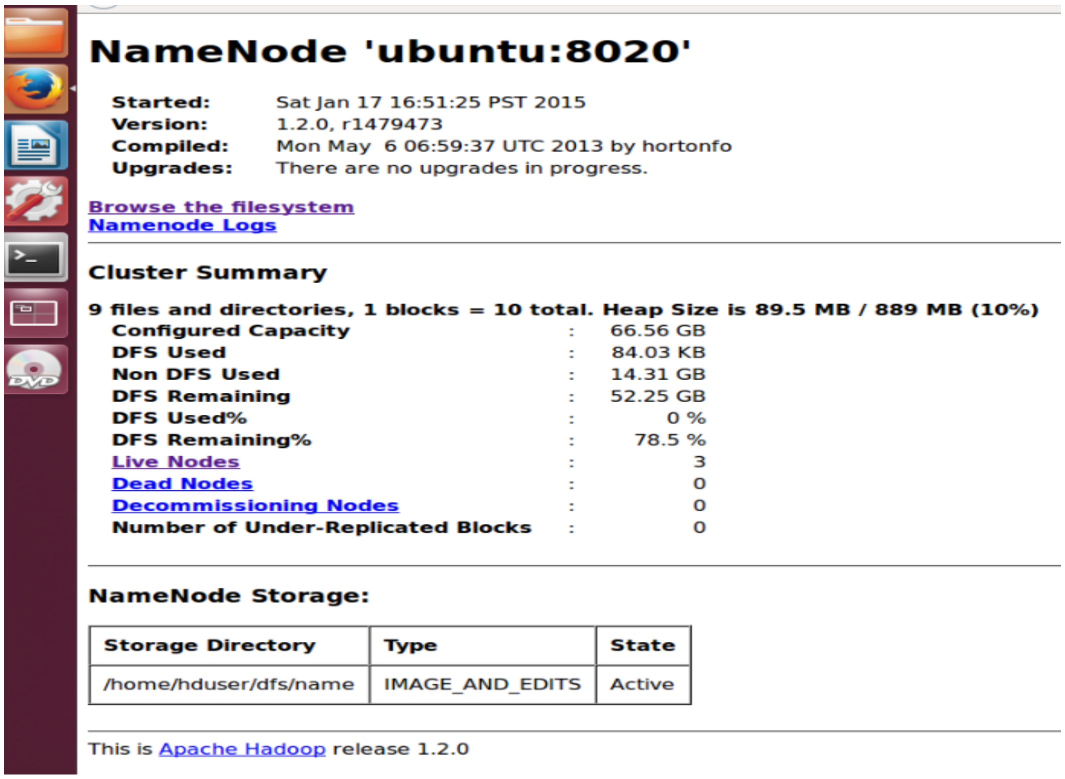
Since this research study is involved in dealing with several distributed framework components, and for choosing a reliable data storage, HDFS is been considered a good choice to source all the ingestion data. A multi 3 node Fully distributed [11] Hadoop 1.2.0 cluster is been created for the research project. The 3 Ubuntu instances are named as 'master11', 'slave11' and 'slave22'. The Ubuntu [35] instance named 'master11' from now on is called as Master node, is configured with both the master and slave daemons of Hadoop. The master daemons of Hadoop are NameNode, Secondary NameNode, JobTracker and the slave daemons of Hadoop are TaskTracker and DataNode. The remaining 2 Ubuntu instances named 'slave11' and 'slave22' from now on are referred as slave nodes. The slave daemons TaskTracker and DataNode are configured on both the slave nodes. Hadoop installation ships with a configuration folder named "conf" which contains few hadoop specific xml files and an environment file. These xml files and the environment file need to be modified with specific properties of each daemon to run Hadoop in a fully distributed mode. The Table

3.4 describes the Hadoop daemons installed on the available instances.

Instance Name	Referred As	Installed Hadoop Daemons
master11	Master Node	NameNode, SecondaryNameNode, JobTracker, DataNode, TaskTracker
slave11	Slave Node 1	DataNode, TaskTracker
slave22	Slave Node 2	DataNode, TaskTracker

Table 3.4. Hadoop Cluster Components

- NameNode: Hadoop HDFS file system administration is done by the NameNode and can be browsed by using NameNode's web addres at "http://Master Node IP:50070/". It also gives a summary of the cluster details about number of live nodes available in the cluster, number of dead nodes, capacity of HDFS configured, capacity of the remaining HDFS left and logs etc. The below Figure 3.2 shows HDFS web interface.
- JobTracker: Hadoop MapReduce administration is taken care by the JobTracker. MapReduce jobs submitted by the client are managed by the JobTracker, and these JobTrackers assign the tasks to the respective available TaskTrackers to process the MapReduce jobs. JobTracker's web address is at "http:://Master Node IP:50030/". JobTracker interface enables us to look at the running jobs, finished jobs, failure jobs and scheduling information etc. User can browse those jobs to get more information about the logs and other core functional features of a MapReduce job. Logs helps us to debug any long running and stale jobs in the cluster.
- dfsadmin: Hadoop provides an admin like file shell command to get the cluster report. The command to get the report is "hadoop dfsadmin -report" [11]. The below Figure 3.5 describes the statistics and health of the Hadoop cluster. It reports the status of each node, configured capacity, percentage of DFS used and remaining, corrupt blocks if any or missing blocks on the node etc.



NameNode 'ubuntu:8020'

Started:	Sat Jan 17 16:51:25 PST 2015
Version:	1.2.0, r1479473
Compiled:	Mon May 6 06:59:37 UTC 2013 by hortonfo
Upgrades:	There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

9 files and directories, 1 blocks = 10 total. Heap Size is 89.5 MB / 889 MB (10%)

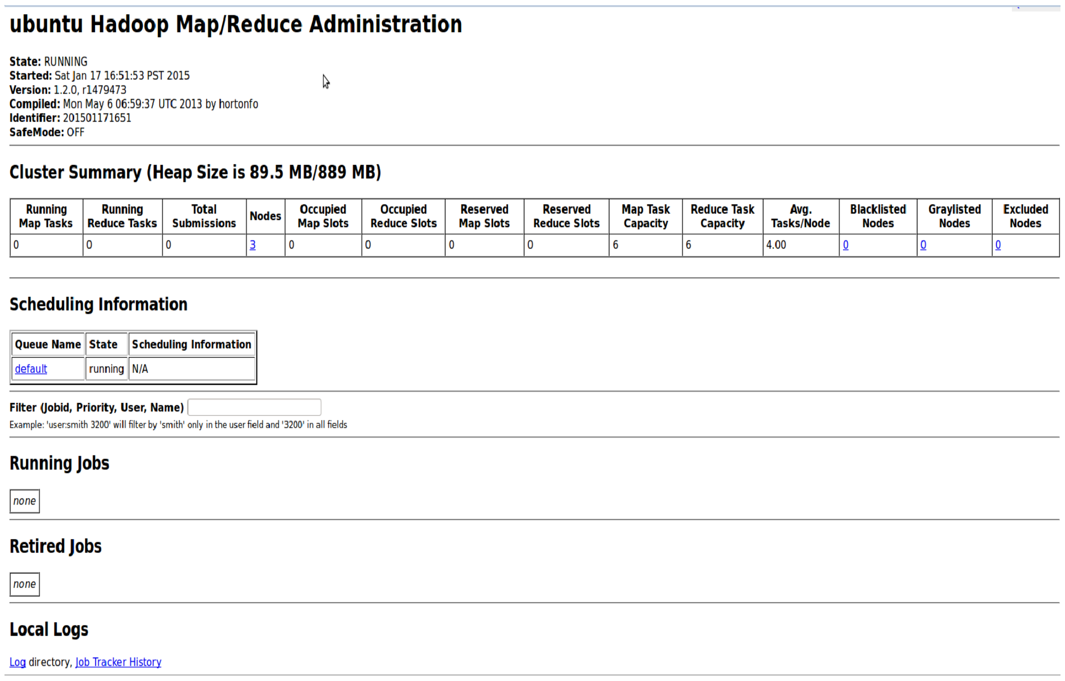
Configured Capacity	:	66.56 GB
DFS Used	:	84.03 KB
Non DFS Used	:	14.31 GB
DFS Remaining	:	52.25 GB
DFS Used%	:	0 %
DFS Remaining%	:	78.5 %
Live Nodes	:	3
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0

NameNode Storage:

Storage Directory	Type	State
/home/hduser/dfs/name	IMAGE_AND_EDITS	Active

This is Apache Hadoop release 1.2.0

Figure 3.2. HDFS NameNode Summary



ubuntu Hadoop Map/Reduce Administration

State: RUNNING
Started: Sat Jan 17 16:51:53 PST 2015
Version: 1.2.0, r1479473
Compiled: Mon May 6 06:59:37 UTC 2013 by hortonfo
Identifier: 201501171651
SafeMode: OFF

Cluster Summary (Heap Size is 89.5 MB/889 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
0	0	0	3	0	0	0	0	6	6	4.00	0	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

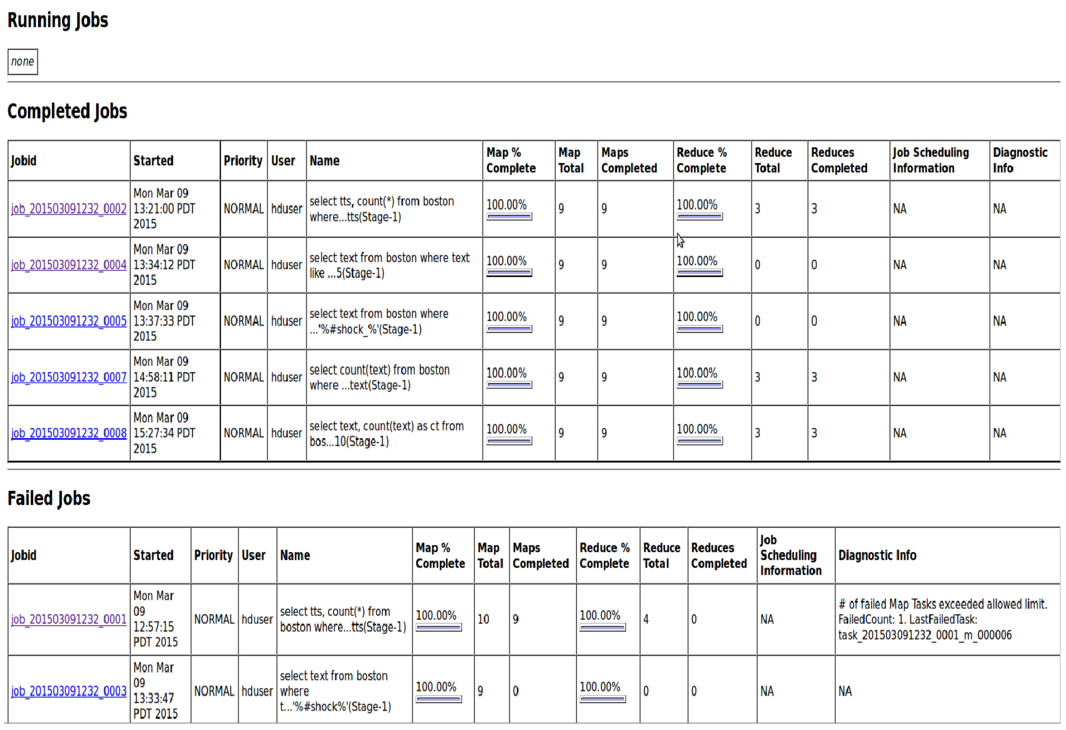
Retired Jobs

none

Local Logs

Log directory, Job Tracker History

Figure 3.3. JobTracker Web Interface Home Page



Running Jobs

none

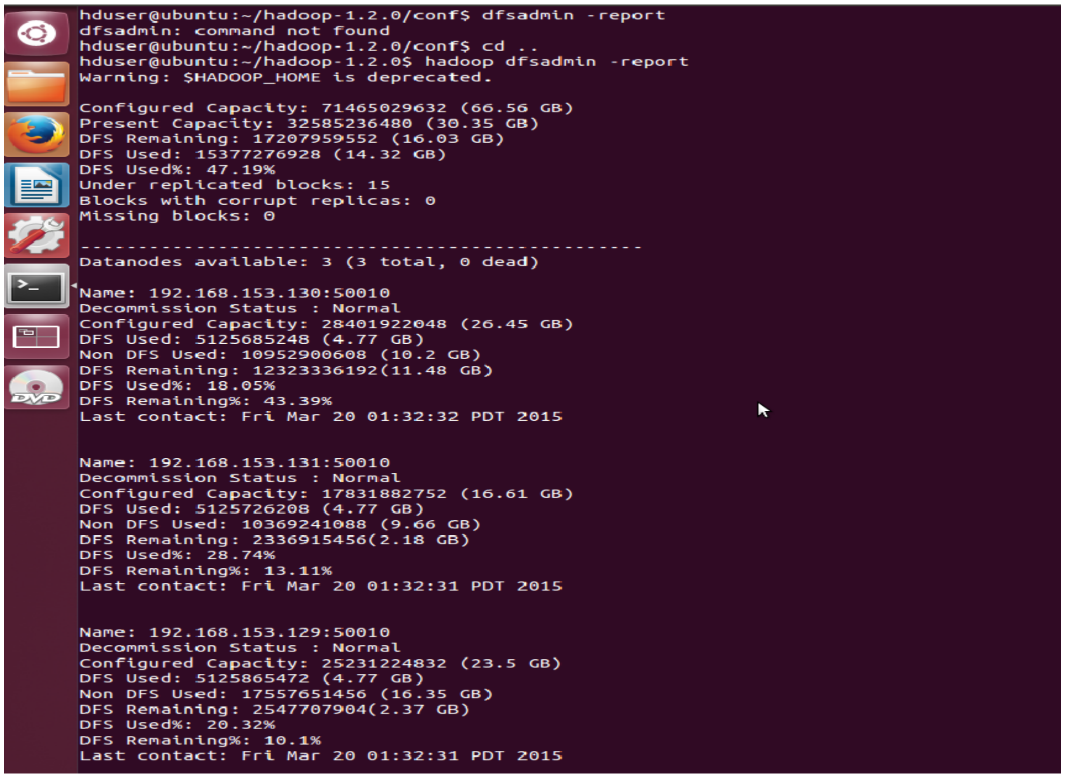
Completed Jobs

Jobid	Started	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201503091232_0002	Mon Mar 09 13:21:00 PDT 2015	NORMAL	hduser	select tts, count(*) from boston where...tts(Stage-1)	100.00%	9	9	100.00%	3	3	NA	NA
job_201503091232_0004	Mon Mar 09 13:34:12 PDT 2015	NORMAL	hduser	select text from boston where text like ...5(Stage-1)	100.00%	9	9	100.00%	0	0	NA	NA
job_201503091232_0005	Mon Mar 09 13:37:33 PDT 2015	NORMAL	hduser	select text from boston where ...'%#shock_%'(Stage-1)	100.00%	9	9	100.00%	0	0	NA	NA
job_201503091232_0007	Mon Mar 09 14:58:11 PDT 2015	NORMAL	hduser	select count(text) from boston where ...text(Stage-1)	100.00%	9	9	100.00%	3	3	NA	NA
job_201503091232_0008	Mon Mar 09 15:27:34 PDT 2015	NORMAL	hduser	select text, count(text) as ct from bos...10(Stage-1)	100.00%	9	9	100.00%	3	3	NA	NA

Failed Jobs

Jobid	Started	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201503091232_0001	Mon Mar 09 12:57:15 PDT 2015	NORMAL	hduser	select tts, count(*) from boston where...tts(Stage-1)	100.00%	10	9	100.00%	4	0	NA	# of failed Map Tasks exceeded allowed limit. FailedCount: 1. LastFailedTask: task_201503091232_0001_m_000006
job_201503091232_0003	Mon Mar 09 13:33:47 PDT 2015	NORMAL	hduser	select text from boston where t...'%#shock%'(Stage-1)	100.00%	9	0	100.00%	0	0	NA	NA

Figure 3.4. JobTracker Jobs Web Interface



```
hduser@ubuntu:~/hadoop-1.2.0/conf$ dfsadmin -report
dfsadmin: command not found
hduser@ubuntu:~/hadoop-1.2.0/conf$ cd ..
hduser@ubuntu:~/hadoop-1.2.0$ hadoop dfsadmin -report
Warning: $HADOOP_HOME is deprecated.

Configured Capacity: 71465029632 (66.56 GB)
Present Capacity: 32585236480 (30.35 GB)
DFS Remaining: 17207959552 (16.03 GB)
DFS Used: 15377276928 (14.32 GB)
DFS Used%: 47.19%
Under replicated blocks: 15
Blocks with corrupt replicas: 0
Missing blocks: 0

-------------------------------------------------
Datanodes available: 3 (3 total, 0 dead)

Name: 192.168.153.130:50010
Decommission Status : Normal
Configured Capacity: 28401922048 (26.45 GB)
DFS Used: 5125685248 (4.77 GB)
Non DFS Used: 10952900608 (10.2 GB)
DFS Remaining: 12323336192(11.48 GB)
DFS Used%: 18.05%
DFS Remaining%: 43.39%
Last contact: Fri Mar 20 01:32:32 PDT 2015


Name: 192.168.153.131:50010
Decommission Status : Normal
Configured Capacity: 17831882752 (16.61 GB)
DFS Used: 5125726208 (4.77 GB)
Non DFS Used: 10369241088 (9.66 GB)
DFS Remaining: 2336915456(2.18 GB)
DFS Used%: 28.74%
DFS Remaining%: 13.11%
Last contact: Fri Mar 20 01:32:31 PDT 2015


Name: 192.168.153.129:50010
Decommission Status : Normal
Configured Capacity: 25231224832 (23.5 GB)
DFS Used: 5125865472 (4.77 GB)
Non DFS Used: 17557651456 (16.35 GB)
DFS Remaining: 2547707904(2.37 GB)
DFS Used%: 20.32%
DFS Remaining%: 10.1%
Last contact: Fri Mar 20 01:32:31 PDT 2015
```

Figure 3.5. Dfsadmin Report

3.2.2. Sqoop

This research study also involves in bringing the data from PostgreSQL 9 Database [25] onto Hadoop. As mentioned earlier from the Background section 2.4.1 Sqoop is a tool that facilitates to bring in the data from Databases. Before working with Sqoop firstly, PostgreSQL Database need to be configured to listen to the IP on where Sqoop client is installed. ‘’pg_hba.conf”, “postgresql.conf” are the configuration files on PostgreSQL that need to be changed to listen to the IP addresses. Secondly, the respective vendor specific JDBC jars that are required for the Sqoop to interact with any Database need to be installed in the “lib” folder of Sqoop. Sqoop can be configured properly on the Master Node by modifying the environment file ‘’sqoop-env.sh” of Sqoop installation. Some environment variables are HADOOP_HOME, HBASE_HOME and HIVE_HOME.

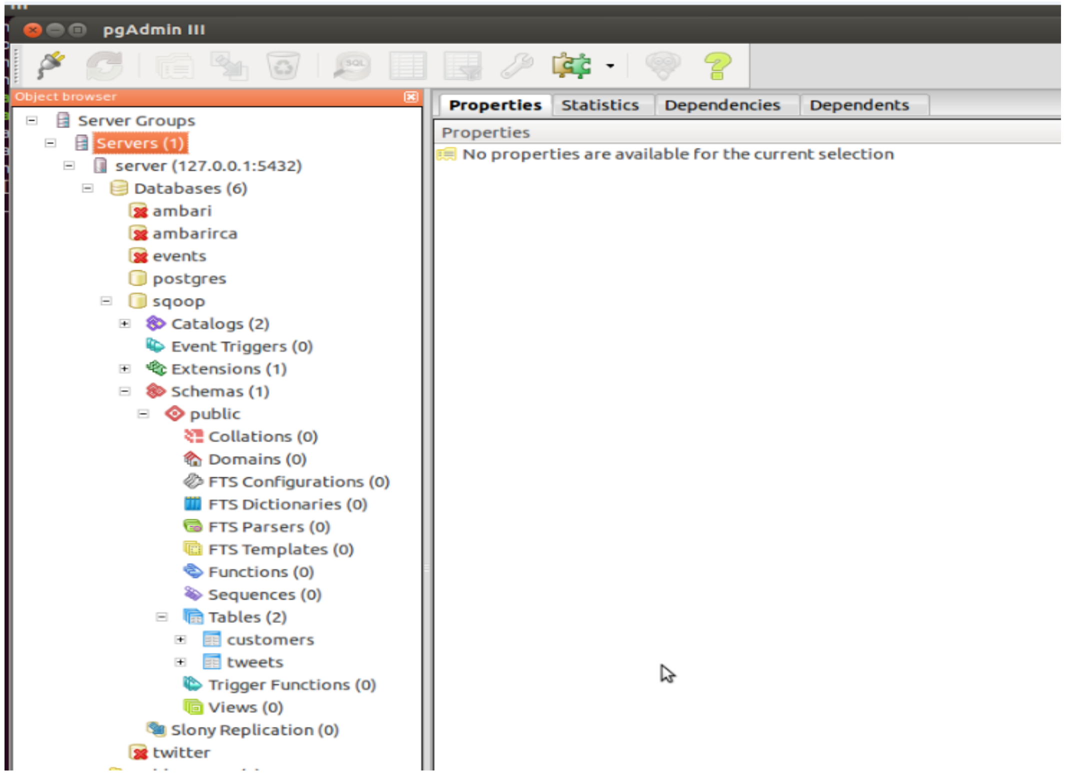
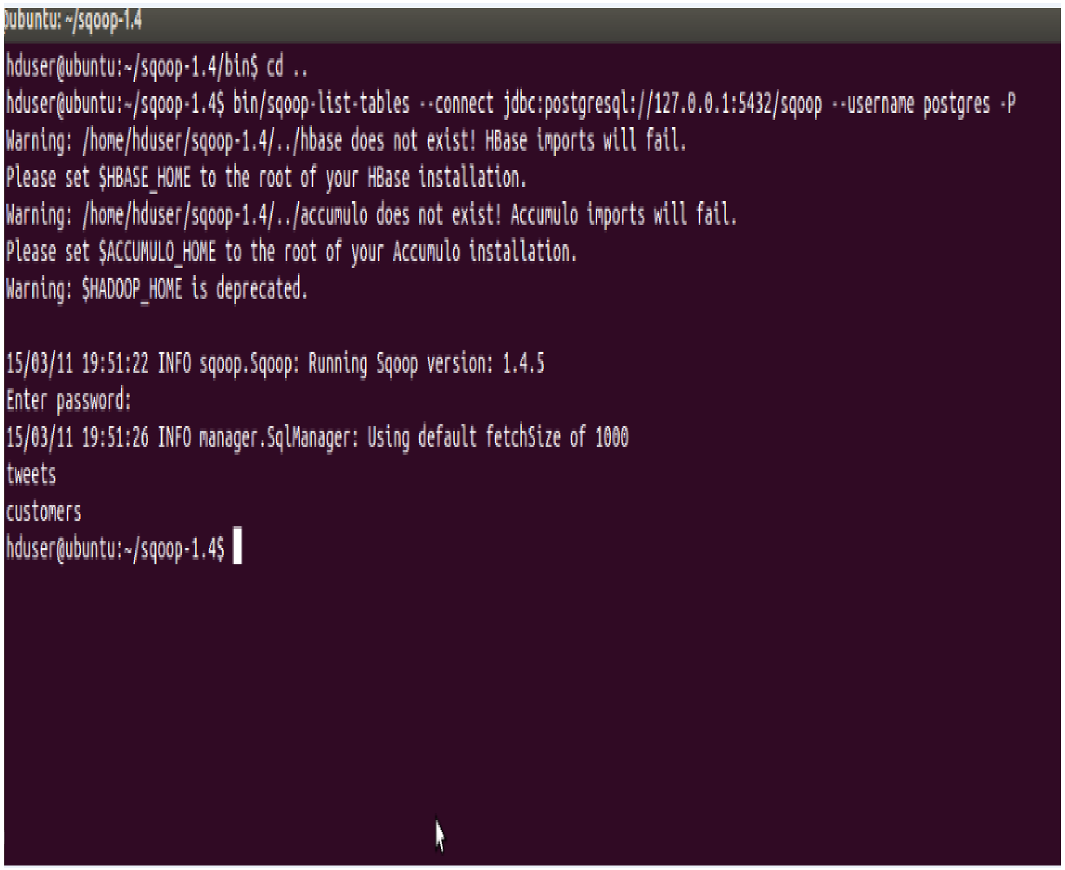


FIGURE 3.6. PostgreSQL Database

The above Figure 3.6 is a PostgreSQL 9 Database created with a database named sqoop and tables named tweets, customers. The Figure 3.7 is a “sqoop-list-tables” job. If the connection from Sqoop is OK, then it will list all the tables in the specified sqoop database. In this case, the job successfully fetches the list of those tables as an output of the job.



```
@ubuntu: ~/sqoop-1.4
hduser@ubuntu:~/sqoop-1.4/bin$ cd ..
hduser@ubuntu:~/sqoop-1.4$ bin/sqoop-list-tables --connect jdbc:postgresql://127.0.0.1:5432/sqoop --username postgres -P
Warning: /home/hduser/sqoop-1.4/../hbase does not exist! HBase imports will fail.
Please set $HBASE_HOME to the root of your HBase installation.
Warning: /home/hduser/sqoop-1.4/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
Warning: $HADOOP_HOME is deprecated.

15/03/11 19:51:22 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5
Enter password:
15/03/11 19:51:26 INFO manager.SqlManager: Using default fetchSize of 1000
tweets
customers
hduser@ubuntu:~/sqoop-1.4$
```

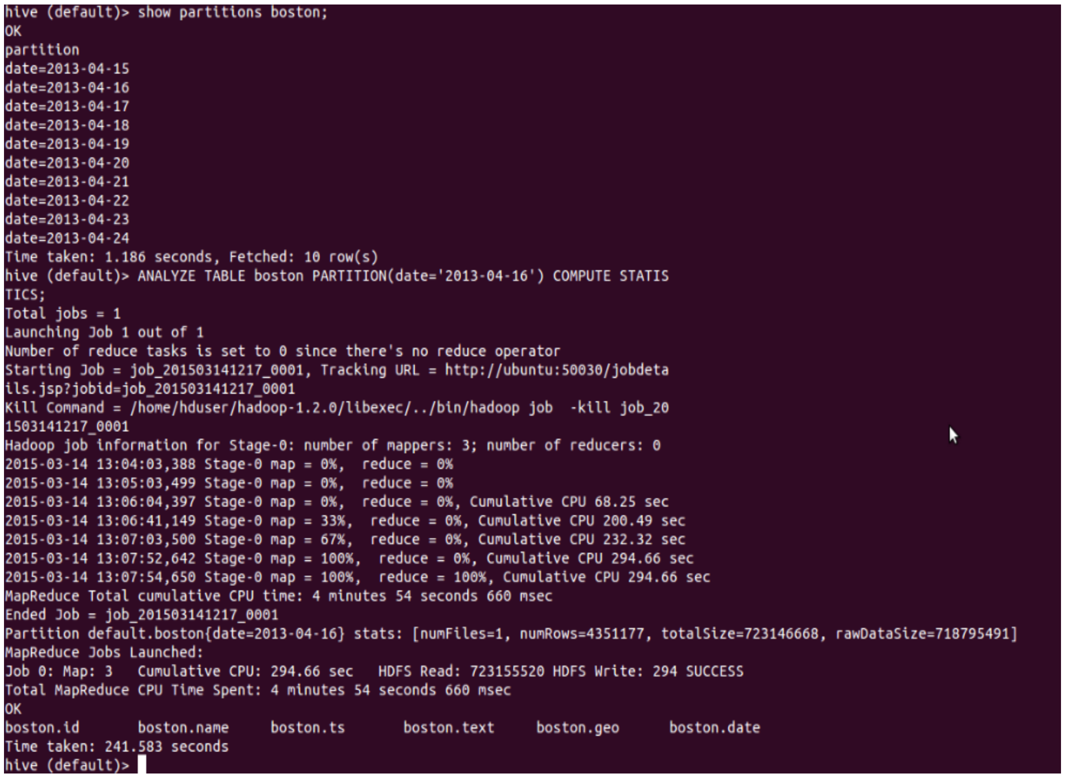
FIGURE 3.7. Sqoop List Tables

3.2.3. Flume

Flume is a tool to bring in streaming data or log data onto target stores. This research study has ingested the Twitter Streaming API data using Flume onto target HDFS store. The detailed description about TwitterAgent, structure of tweets ingested, HDFS storage area are shown as diagrams in the later Chapter 4.

3.2.4. Hive

The structured text data using Sqoop and semi structured json data ingested using Flume on HDFS is been processed on Hive for analytics. Hive is considered in this research because of its ability to store and process this kind of json data by supporting both primitive and collection data types. The flexibility of using Hive and the efficiency of it are shown in the usecase at Chapter 4. The "conf" folder of Hive distribution ships with few configurtation xml files and an environment file. These environment variables need to be configured properly.



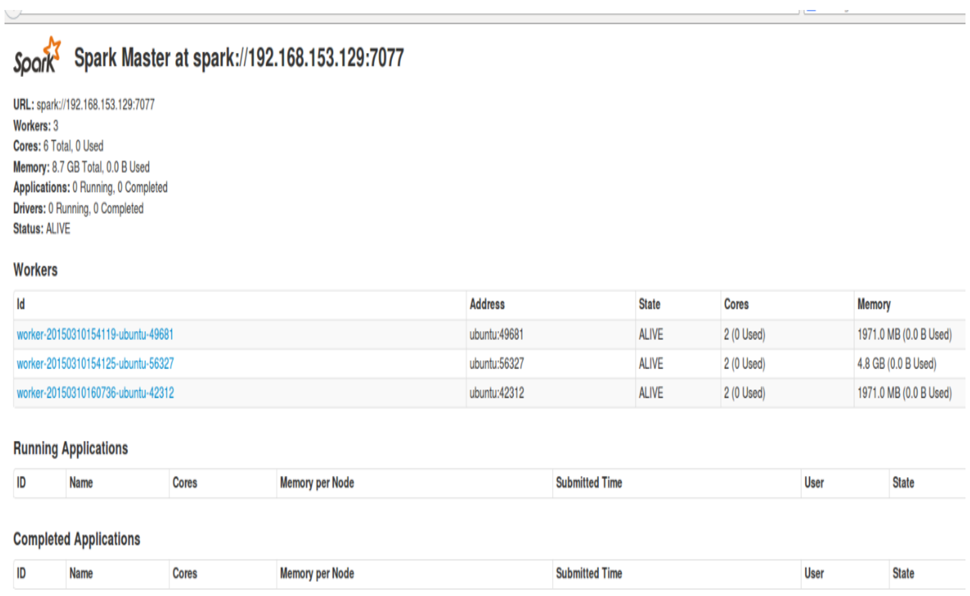
```
hive (default)> show partitions boston;
OK
partition
date=2013-04-15
date=2013-04-16
date=2013-04-17
date=2013-04-18
date=2013-04-19
date=2013-04-20
date=2013-04-21
date=2013-04-22
date=2013-04-23
date=2013-04-24
Time taken: 1.186 seconds, Fetched: 10 row(s)
hive (default)> ANALYZE TABLE boston PARTITION(date='2013-04-16') COMPUTE STATIS
TICS;
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201503141217_0001, Tracking URL = http://ubuntu:50030/jobdeta
ils.jsp?jobid=job_201503141217_0001
Kill Command = /home/hduser/hadoop-1.2.0/libexec/../bin/hadoop job  -kill job_20
1503141217_0001
Hadoop job information for Stage-0: number of mappers: 3; number of reducers: 0
2015-03-14 13:04:03,388 Stage-0 map = 0%,  reduce = 0%
2015-03-14 13:05:03,499 Stage-0 map = 0%,  reduce = 0%
2015-03-14 13:06:04,397 Stage-0 map = 0%,  reduce = 0%, Cumulative CPU 68.25 sec
2015-03-14 13:06:41,149 Stage-0 map = 33%,  reduce = 0%, Cumulative CPU 200.49 sec
2015-03-14 13:07:03,500 Stage-0 map = 67%,  reduce = 0%, Cumulative CPU 232.32 sec
2015-03-14 13:07:52,642 Stage-0 map = 100%,  reduce = 0%, Cumulative CPU 294.66 sec
2015-03-14 13:07:54,650 Stage-0 map = 100%,  reduce = 100%, Cumulative CPU 294.66 sec
MapReduce Total cumulative CPU time: 4 minutes 54 seconds 660 msec
Ended Job = job_201503141217_0001
Partition default.boston{date=2013-04-16} stats: [numFiles=1, numRows=4351177, totalSize=723146668, rawDataSize=718795491]
MapReduce Jobs Launched:
Job 0: Map: 3   Cumulative CPU: 294.66 sec   HDFS Read: 723155520 HDFS Write: 294 SUCCESS
Total MapReduce CPU Time Spent: 4 minutes 54 seconds 660 msec
OK
boston.id       boston.name     boston.ts       boston.text     boston.geo      boston.date
Time taken: 241.583 seconds
hive (default)>
```

FIGURE 3.8. Hive CLI

The above Figure 3.8 is a "Hive Command Line Interface" (Hive CLI) [1, 13]. Hive CLI is a command line service interaction to Hive. HiveQL is a SQl like query interface to run queries on Hive.

3.2.5. Spark

The part of this research study also explores the uses of Spark when working with the distributed systems. Spark enables the users to work with streaming data as well as structured data. Spark also extends the compatability to use the data sourced on Hive. For this research, "Spark standalone mode" [29] is deployed to work along with Hadoop. SparkSQL, a SQL like module is considered to query the data sourced in Hive.



Spark Master at spark://192.168.153.129:7077

URL: spark://192.168.153.129:7077
Workers: 3
Cores: 6 Total, 0 Used
Memory: 8.7 GB Total, 0.0 B Used
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers

Id	Address	State	Cores	Memory
worker-20150310154119-ubuntu-49681	ubuntu:49681	ALIVE	2 (0 Used)	1971.0 MB (0.0 B Used)
worker-20150310154125-ubuntu-56327	ubuntu:56327	ALIVE	2 (0 Used)	4.8 GB (0.0 B Used)
worker-20150310160736-ubuntu-42312	ubuntu:42312	ALIVE	2 (0 Used)	1971.0 MB (0.0 B Used)

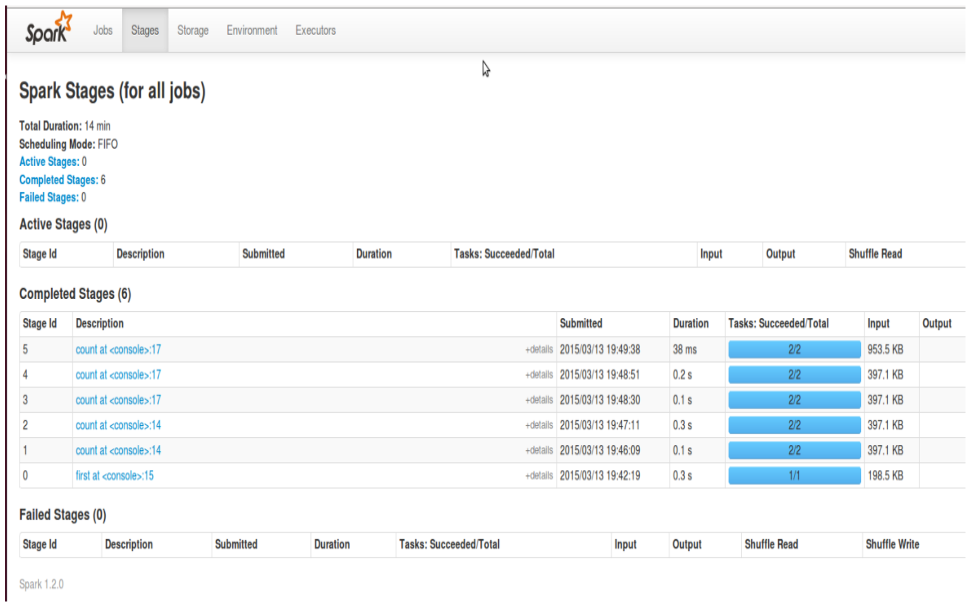
Running Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State

Completed Applications

ID	Name	Cores	Memory per Node	Submitted Time	User	State

FIGURE 3.9. Spark Standalone Cluster Home Page



Spark Stages (for all jobs)

Total Duration: 14 min
Scheduling Mode: FIFO
Active Stages: 0
Completed Stages: 6
Failed Stages: 0

Active Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read

Completed Stages (6)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output
5	count at <console>:17	+details	2015/03/13 19:49:38	38 ms	2/2	953.5 KB	
4	count at <console>:17	+details	2015/03/13 19:48:51	0.2 s	2/2	397.1 KB	
3	count at <console>:17	+details	2015/03/13 19:48:30	0.1 s	2/2	397.1 KB	
2	count at <console>:14	+details	2015/03/13 19:47:11	0.3 s	2/2	397.1 KB	
1	count at <console>:14	+details	2015/03/13 19:46:09	0.1 s	2/2	397.1 KB	
0	first at <console>:15	+details	2015/03/13 19:42:19	0.3 s	1/1	198.5 KB	

Failed Stages (0)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write

Spark 1.2.0

FIGURE 3.10. Spark Standalone Cluster Jobs for Count Queries

The Figure 3.9 is a 3 Node Standalone Spark Cluster with one Node serving as both Master, Slave and the remaining two as only Slaves. Spark clusters need to have Driver Mangers [42] to manage and can be installed in Standalone Mode, or YARN or MESOS. This research is only limited to Standalone Mode [15]

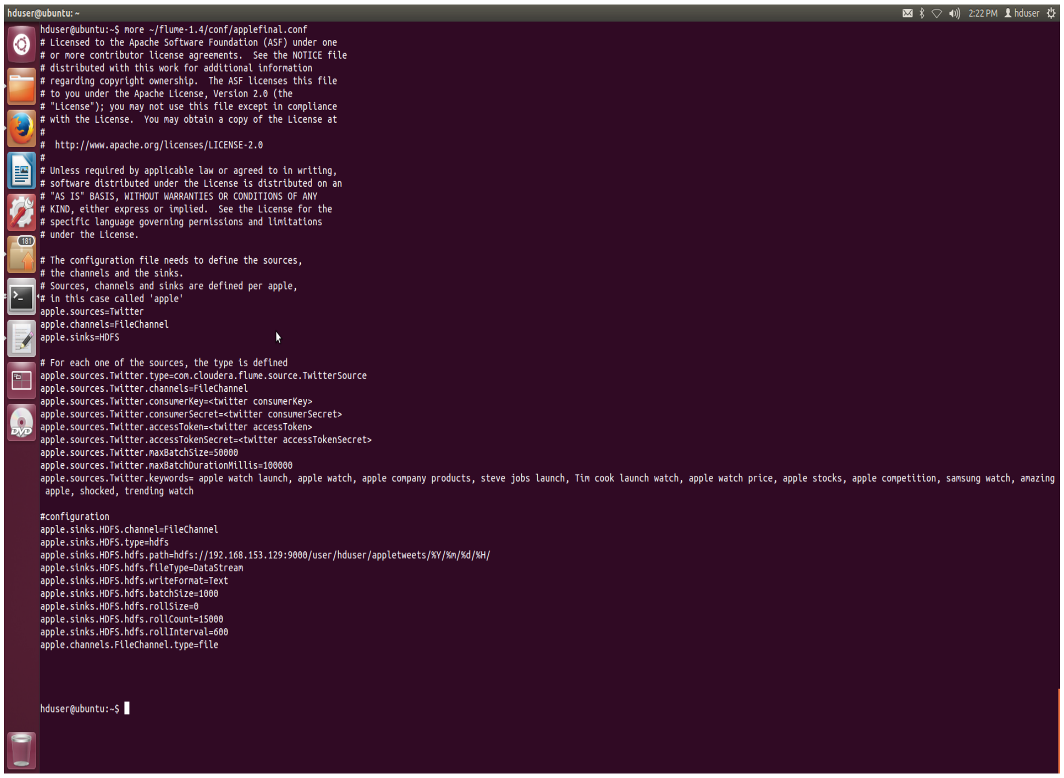
CHAPTER 4

USE CASE AND RESULTS

This research study has taken a use case to “Analyze the Twitter Streaming API data” [3] the tweets during the release of Iphone watch. The frameworks used for the analyis are Flume, Hive and Spark. The streaming data is pulled using Flume and sourced onto target HDFS store for storage. The semi structured json data is interpreted on Hive and Spark for the end user analytics. This use case study is much benefitted from the cloudera cdh-twitter- example [3].

4.1. Configuring Flume

Flume can bring in the events using Source, pass them on to Channel and finally will be consumed by Sinks [6]. All these components need to be configured in a Flume agent. The Figure 4.1 [19, 6] defines the Flume Agent named apple.



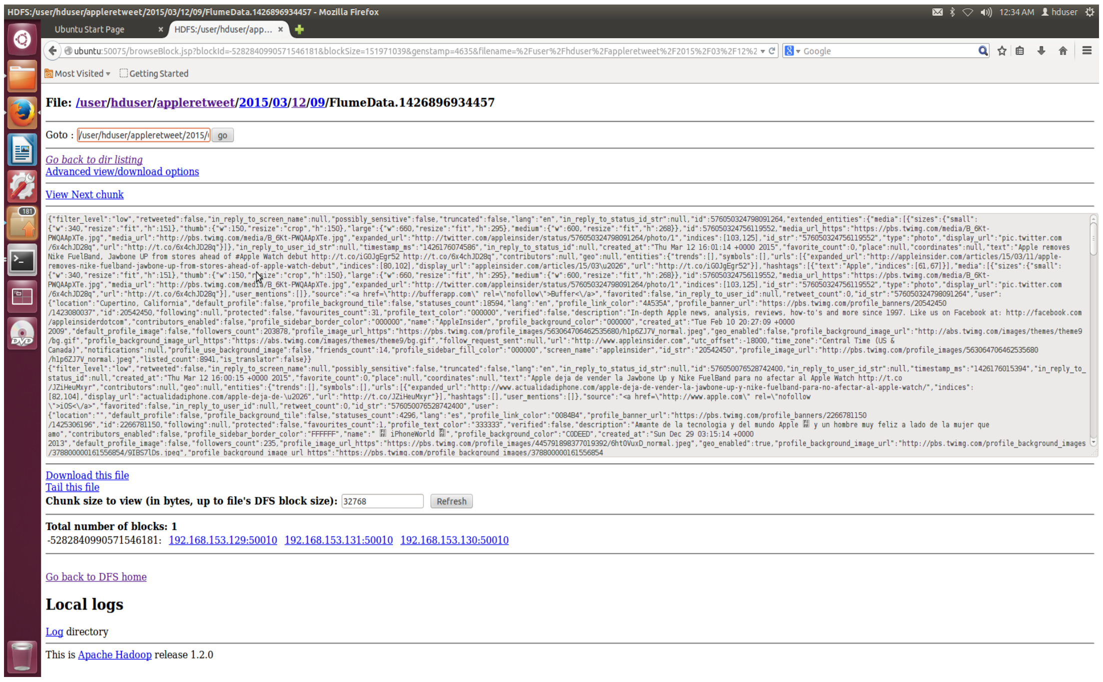
```
hduser@ubuntu:~$ more ~/flume-1.4/conf/applefinal.conf
# Licensed to the Apache Software Foundation (ASF) under one
# or more contributor license agreements.  See the NOTICE file
# distributed with this work for additional information
# regarding copyright ownership.  The ASF licenses this file
# to you under the Apache License, Version 2.0 (the
# "License"); you may not use this file except in compliance
# with the License.  You may obtain a copy of the License at
#
#  http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
# software distributed under the License is distributed on an
# "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
# KIND, either express or implied.  See the License for the
# specific language governing permissions and limitations
# under the License.

# The configuration file needs to define the sources,
# the channels and the sinks.
# Sources, channels and sinks are defined per apple,
# in this case called 'apple'
apple.sources=Twitter
apple.channels=FileChannel
apple.sinks=HDFS

# For each one of the sources, the type is defined
apple.sources.Twitter.type=com.cloudera.flume.source.TwitterSource
apple.sources.Twitter.channels=FileChannel
apple.sources.Twitter.consumerKey=<twitter consumerKey>
apple.sources.Twitter.consumerSecret=<twitter consumerSecret>
apple.sources.Twitter.accessToken=<twitter accessToken>
apple.sources.Twitter.accessTokenSecret=<twitter accessTokenSecret>
apple.sources.Twitter.maxBatchSize=50000
apple.sources.Twitter.maxBatchDurationMillis=100000
apple.sources.Twitter.keywords= apple watch launch, apple watch, apple company products, steve jobs launch, Tim cook launch watch, apple watch price, apple stocks, apple competition, samsung watch, amazing
 apple, shocked, trending watch

#configuration
apple.sinks.HDFS.channel=FileChannel
apple.sinks.HDFS.type=hdfs
apple.sinks.HDFS.hdfs.path=hdfs://192.168.153.129:9000/user/hduser/appletweets/%Y/%m/%d/%H/
apple.sinks.HDFS.hdfs.fileType=DataStream
apple.sinks.HDFS.hdfs.writeFormat=Text
apple.sinks.HDFS.hdfs.batchSize=1000
apple.sinks.HDFS.hdfs.rollSize=0
apple.sinks.HDFS.hdfs.rollCount=15000
apple.sinks.HDFS.hdfs.rollInterval=600
apple.channels.FileChannel.type=file



hduser@ubuntu:~$
```

Figure 4.1. Flume Agent

Its configured with the Source as Twitter. Flume allows to build a Custom Source or use the exisiting sources of Flume. For this research a custom Flume Source available at [3] is used. The source jar need to be uploaded to the "lib" folder of flume and the path of this jar need to be included for the environment variable "FLUME_CLASSPATH", so that flume agent can recognize it during the start up of the flume job. The Channel considered in this study is a File Channel and Sink is the HDFS location. The Flume agent is configured to search for the keywords ranging like apple watch, apple price, apple stocks, apple competition and apple launch etc were taken. The HDFS location in the Flume agent is configured as "hdfs://192.168.153.129:9000/user/hduser/appletweets/%Y/%m/%d/%H/" [19] so, the tweets gets stored in the HDFS with directory structure like Year, Month, Date, Hour. That is, an event of tweet generated at 09/03/2015 8 00 AM will land in "hdfs://<HDFS LOCATION>/2015/03/09/08." [19]. This format will be useful to make end analytics at more granular level in the form of Partitioning in Hive.

4.2. Data

The Figure 4.2 shows the sample chunk of the Tweet Data retrieved using Flume.



HDFS:/user/hduser/appleretweet/2015/03/12/09/FlumeData.1426896934457 - Mozilla Firefox

ubuntu:50075/browseBlock.jsp?blockId=-5282840990571546181&blockSize=151971039&genstamp=4635&filename=%2Fuser%2Fhduser%2Fappleretweet%2F2015%2F03%2F12%2

File: /user/hduser/appleretweet/2015/03/12/09/FlumeData.1426896934457

Goto : /user/hduser/appleretweet/2015/

Go back to dir listing
Advanced view/download options

View Next chunk

```
{"filter_level":"low","retweeted":false,"in_reply_to_screen_name":null,"possibly_sensitive":false,"truncated":false,"lang":"en","in_reply_to_status_id_str":null,"id":576050324798091264,"extended_entities":{"media":[{"sizes":{"small":
{"w":340,"resize":"fit","h":151},"thumb":{"w":150,"resize":"crop","h":150},"large":{"w":660,"resize":"fit","h":295},"medium":{"w":600,"resize":"fit","h":268}},"id":576050324756119552,"media_url_https":"https://pbs.twimg.com/media/B_6Kt-
PWQAApXTe.jpg","media_url":"http://pbs.twimg.com/media/B_6Kt-PWQAApXTe.jpg","expanded_url":"http://twitter.com/appleinsider/status/576050324798091264/photo/1","indices":[103,125],"id_str":"576050324756119552","type":"photo","display_url":"pic.twitter.com
/6x4chJD28q","url":"http://t.co/6x4chJD28q"}]},"in_reply_to_user_id_str":null,"timestamp_ms":"1426176074586","in_reply_to_status_id":null,"created_at":"Thu Mar 12 16:01:14 +0000 2015","favorite_count":0,"place":null,"coordinates":null,"text":"Apple removes
Nike FuelBand, Jawbone UP from stores ahead of #Apple Watch debut http://t.co/iG0JgEgr52 http://t.co/6x4chJD28q","contributors":null,"geo":null,"entities":{"trends":[],"symbols":[],"urls":[{"expanded_url":"http://appleinsider.com/articles/15/03/11/apple-
removes-nike-fuelband-jawbone-up-from-stores-ahead-of-apple-watch-debut","indices":[80,102],"display_url":"appleinsider.com/articles/15/03\u2026","url":"http://t.co/iG0JgEgr52"}],"hashtags":[{"text":"Apple","indices":[61,67]}],"media":[{"sizes":{"small":
{"w":340,"resize":"fit","h":151},"thumb":{"w":150,"resize":"crop","h":150},"large":{"w":660,"resize":"fit","h":295},"medium":{"w":600,"resize":"fit","h":268}},"id":576050324756119552,"media_url_https":"https://pbs.twimg.com/media/B_6Kt-
PWQAApXTe.jpg","media_url":"http://pbs.twimg.com/media/B_6Kt-PWQAApXTe.jpg","expanded_url":"http://twitter.com/appleinsider/status/576050324798091264/photo/1","indices":[103,125],"id_str":"576050324756119552","type":"photo","display_url":"pic.twitter.com
/6x4chJD28q","url":"http://t.co/6x4chJD28q"}],"user_mentions":[]},"source":"<a href=\"http://bufferapp.com\" rel=\"nofollow\">Buffer<\/a>","favorited":false,"in_reply_to_user_id":null,"retweet_count":0,"id_str":"576050324798091264","user":
{"location":"Cupertino, California","default_profile":false,"profile_background_tile":false,"statuses_count":18594,"lang":"en","profile_link_color":"4A535A","profile_banner_url":"https://pbs.twimg.com/profile_banners/20542450
/1423080037","id":20542450,"following":null,"protected":false,"favourites_count":31,"profile_text_color":"000000","verified":false,"description":"In-depth Apple news, analysis, reviews, how-to's and more since 1997. Like us on Facebook at: http://facebook.com
/appleinsiderdotcom","contributors_enabled":false,"profile_sidebar_border_color":"000000","name":"AppleInsider","profile_background_color":"000000","created_at":"Tue Feb 10 20:27:09 +0000
2009","default_profile_image":false,"followers_count":203878,"profile_image_url_https":"https://pbs.twimg.com/profile_images/563064706462535680/h1p6ZJ7V_normal.jpeg","geo_enabled":false,"profile_background_image_url":"http://abs.twimg.com/images/themes/theme9
/bg.gif","profile_background_image_url_https":"https://abs.twimg.com/images/themes/theme9/bg.gif","follow_request_sent":null,"url":"http://www.appleinsider.com","utc_offset":-18000,"time_zone":"Central Time (US &
Canada)","notifications":null,"profile_use_background_image":false,"friends_count":14,"profile_sidebar_fill_color":"000000","screen_name":"appleinsider","id_str":"20542450","profile_image_url":"http://pbs.twimg.com/profile_images/563064706462535680
/h1p6ZJ7V_normal.jpeg","listed_count":8941,"is_translator":false}}
{"filter_level":"low","retweeted":false,"in_reply_to_screen_name":null,"possibly_sensitive":false,"truncated":false,"lang":"en","in_reply_to_status_id_str":null,"id":576050076528742400,"in_reply_to_user_id_str":null,"timestamp_ms":"1426176015394","in_reply_to_
status_id":null,"created_at":"Thu Mar 12 16:00:15 +0000 2015","favorite_count":0,"place":null,"coordinates":null,"text":"Apple deja de vender la Jawbone Up y Nike FuelBand para no afectar al Apple Watch http://t.co
/JZiHeuMxyr","contributors":null,"geo":null,"entities":{"trends":[],"symbols":[],"urls":[{"expanded_url":"http://www.actualidadiphone.com/apple-deja-de-vender-la-jawbone-up-y-nike-fuelband-para-no-afectar-al-apple-watch/","indices":
[82,104],"display_url":"actualidadiphone.com/apple-deja-de-\u2026","url":"http://t.co/JZiHeuMxyr"}],"hashtags":[],"user_mentions":[]},"source":"<a href=\"http://www.apple.com\" rel=\"nofollow
\">iOS<\/a>","favorited":false,"in_reply_to_user_id":null,"retweet_count":0,"id_str":"576050076528742400","user":
{"location":"","default_profile":false,"profile_background_tile":false,"statuses_count":4296,"lang":"es","profile_link_color":"0084B4","profile_banner_url":"https://pbs.twimg.com/profile_banners/2266781150
/1425306196","id":2266781150,"following":null,"protected":false,"favourites_count":1,"profile_text_color":"333333","verified":false,"description":"Amante de la tecnologia y del mundo Apple □ y un hombre muy feliz a lado de la mujer que
amo","contributors_enabled":false,"profile_sidebar_border_color":"FFFFFF","name":" □ iPhoneWorld □","profile_background_color":"C0DEED","created_at":"Sun Dec 29 03:15:14 +0000
2013","default_profile_image":false,"followers_count":235,"profile_image_url_https":"https://pbs.twimg.com/profile_images/445791898377019392/6ht0VuxD_normal.jpeg","geo_enabled":true,"profile_background_image_url":"http://pbs.twimg.com/profile_background_images
/378800000161556854/9IBS7lDs.jpeg","profile_background_image_url_https":"https://pbs.twimg.com/profile_background_images/378800000161556854
```

Download this file
Tail this file
Chunk size to view (in bytes, up to file's DFS block size): 32768 Refresh

Total number of blocks: 1
-5282840990571546181: 192.168.153.129:50010 192.168.153.131:50010 192.168.153.130:50010

Go back to DFS home

Local logs

Log directory

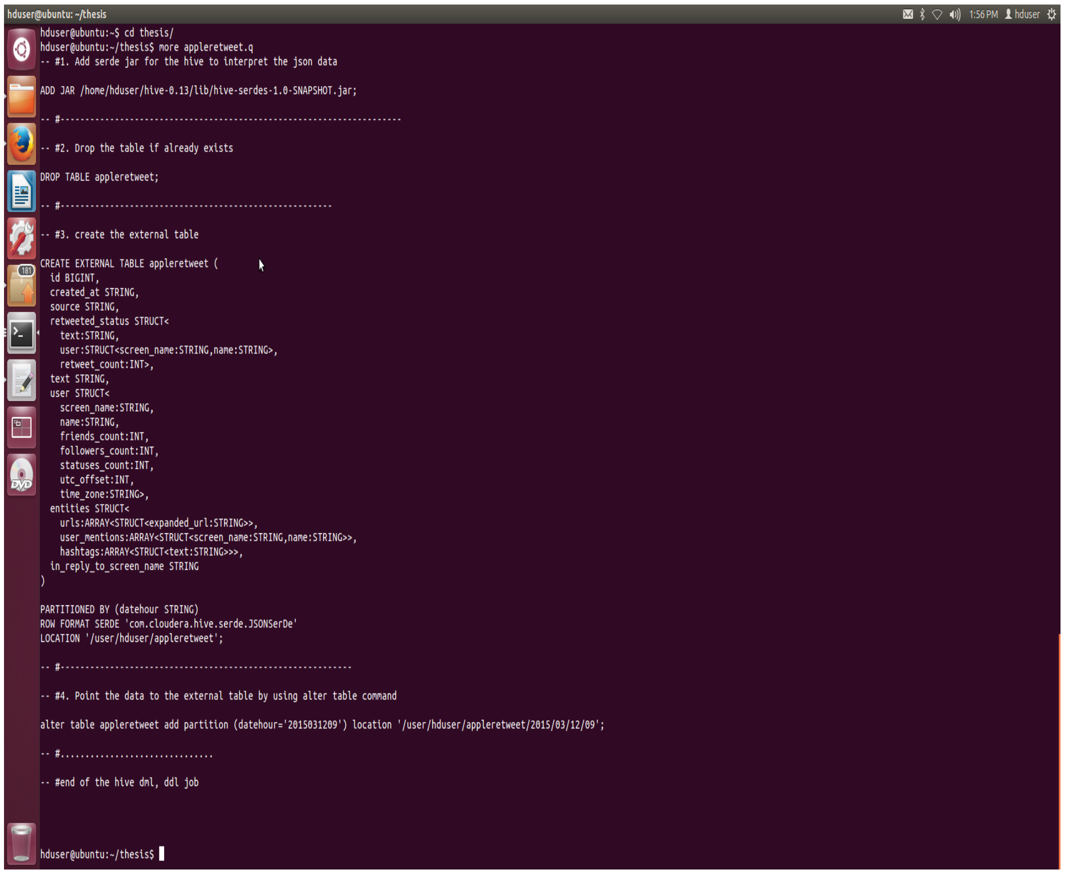
This is Apache Hadoop release 1.2.0

FIGURE 4.2. Flume Tweet Json Data

The Data accumulated using Flume is around 1.4 GB and is in json structure.

4.3. Configuring Hive

Tweet data provide lot of user information and is used for analysis by many companies in the form of marketing their products and providing better services. The Twitter data brought in using Flume need to be analyzed and Hive is considered as an option in this study because of its support to interpret semi structured data by providing complex collection data types like "ARRAY, MAP, STRUCT" [38, 1] for a better table design to access the data efficiently.



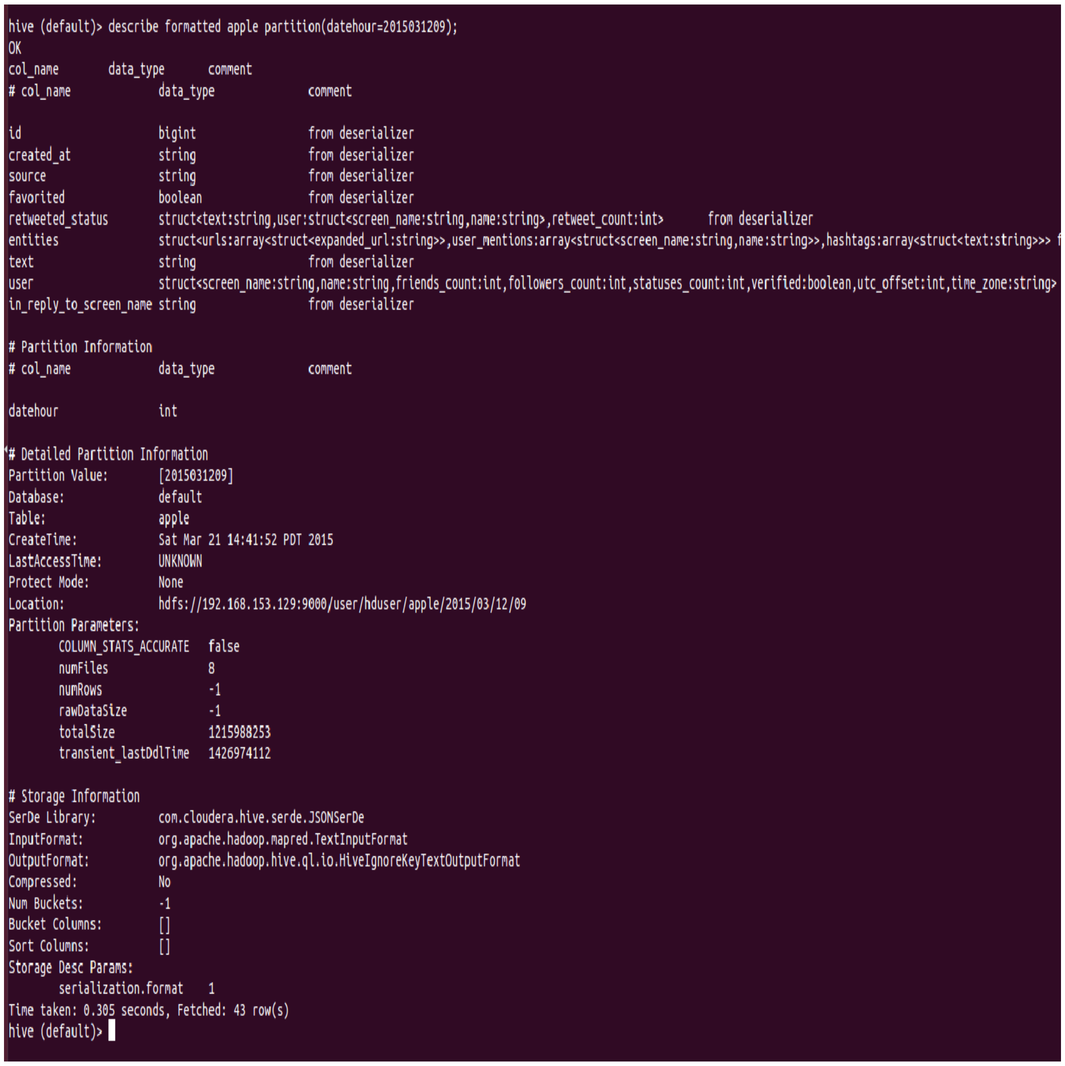
```
hduser@ubuntu:~$ cd thesis/
hduser@ubuntu:~/thesis$ more appleretweet.q
-- #1. Add serde jar for the hive to interpret the json data

ADD JAR /home/hduser/hive-0.13/lib/hive-serdes-1.0-SNAPSHOT.jar;

-- #-----------------------------------------------------------------------

-- #2. Drop the table if already exists

DROP TABLE appleretweet;

-- #-----------------------------------------------------------

-- #3. create the external table

CREATE EXTERNAL TABLE appleretweet (
  id BIGINT,
  created_at STRING,
  source STRING,
  retweeted_status STRUCT<
    text:STRING,
    user:STRUCT<screen_name:STRING,name:STRING>,
    retweet_count:INT>,
  text STRING,
  user STRUCT<
    screen_name:STRING,
    name:STRING,
    friends_count:INT,
    followers_count:INT,
    statuses_count:INT,
    utc_offset:INT,
    time_zone:STRING>,
  entities STRUCT<
    urls:ARRAY<STRUCT<expanded_url:STRING>>,
    user_mentions:ARRAY<STRUCT<screen_name:STRING,name:STRING>>,
    hashtags:ARRAY<STRUCT<text:STRING>>>,
  in_reply_to_screen_name STRING
)

PARTITIONED BY (datehour STRING)
ROW FORMAT SERDE 'com.cloudera.hive.serde.JSONSerDe'
LOCATION '/user/hduser/appleretweet';

-- #-------------------------------------------------------------

-- #4. Point the data to the external table by using alter table command

alter table appleretweet add partition (datehour='2015031209') location '/user/hduser/appleretweet/2015/03/12/09';

-- #..............................

-- #end of the hive dml, ddl job



hduser@ubuntu:~/thesis$
```

FIGURE 4.3. External Tweets Table

The above Figure 4.3 [3, 19] is an External Table structure created in Hive for the Json Tweet data. External Table have few advantages as discussed earlier in 2.4.3.1. In this case, as the Data is ingesting through Flume agent into the HDFS location, this External

Table can also be pointed to the same HDFS location using "LOCATION" [1] keyword in the Table definiton. If a Managed Table is used here for the Table design, then Hive moves the data from the HDFS location to its warehouse directory during the ''LOAD" [1, 38] operation and when the Table is dropped then entire data will also be lost. This is where External Table design comes in a good use because, the data still persists even when the Table is dropped. The Data for the External Table now resides in HDFS location shown in Figure 4.2 specified by the Flume agent, and as the Table is defined as External there is no need to move the data into the Table. However, as the data is ingesting into the HDFS there is a need to add the partitions to have the view of the latest Data in Hive. Tables can be Partitioned on a column by defining "PARTITION BY" [1, 5] clause during the Table definition. By using Partitioning technique, Hive can manage the data into chunks of directories based on the Partition column data type. In this case, the External Table is Partitioned on the "datehour" column with the data type STRING. This hourly Data partition can be added into Hive by using the DML statement "ALTER TABLE" [1] which provides access to the latest hourly Data required for the Hive to do some analytics.

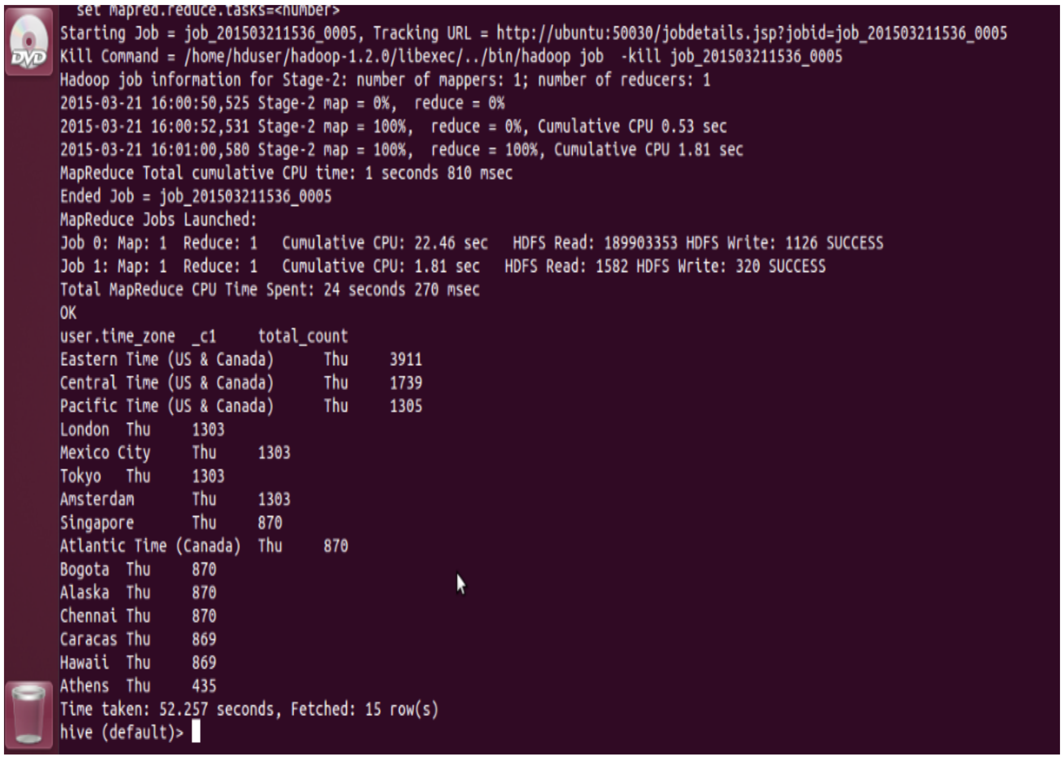
InputFormat and OutputFormat in a MapReduce job specifies the format of the key-value pairs to be read from and write to the files. Hive uses "TextInputFormat and HiveIgnoreKeyTextOutputFormat" [10, 1] as the Input and Output file formats. "Hive stores the Table data as a TEXTFILE by default" [1]. That is, Hive doesnot require any explicit declaration of "STORED AS TEXTFILE" term when creating the table structure. In this usecase the data pulled from Twitter API is a semi structured Json data but Hive support TEXTFILE as a default storage. This Json data can be interpreted by Hive using the Json SerDe [1, 10]. "SerDe is called serializer, deserializer" [1]. SerDes will be useful for Hive to parse the unstructured data in files as records. Hive has few built in SerDes. The details about SerDe is mentioned at 2.4.3.2. This study has used the Json SerDe from [3]. If Hive require any type of SerDes like CSV, Regex or Avro etc., they need to be placed in the "lib" folder of Hive installation and the jars are added during the Hive CLI session using "ADD JAR <path-to-serde>" [1] command. Hive can parse the Json Tweet data by using the “ROW FORMAT SERDE “ [1, 10] clause specified during the Table definition. The Data is now available for the External Table with Partitions on the datehour column. The Figure 4.4 is a detailed formatted description of the Table partition on datehour=’2015031209’ for the External Table “apple”. The total size on that partition is specified as 1.2 GB. The Partition also describes the SerDe used, HDFS location, Partition column used and number of files in the Partition etc.



```
hive (default)> describe formatted apple partition(datehour=2015031209);
OK
col_name        data_type       comment
# col_name              data_type               comment

id                      bigint                  from deserializer
created_at              string                  from deserializer
source                  string                  from deserializer
favorited               boolean                 from deserializer
retweeted_status        struct<text:string,user:struct<screen_name:string,name:string>,retweet_count:int>       from deserializer
entities                struct<urls:array<struct<expanded_url:string>>,user_mentions:array<struct<screen_name:string,name:string>>,hashtags:array<struct<text:string>>> f
text                    string                  from deserializer
user                    struct<screen_name:string,name:string,friends_count:int,followers_count:int,statuses_count:int,verified:boolean,utc_offset:int,time_zone:string>
in_reply_to_screen_name string                  from deserializer

# Partition Information
# col_name              data_type               comment

datehour                int

# Detailed Partition Information
Partition Value:        [2015031209]
Database:               default
Table:                  apple
CreateTime:             Sat Mar 21 14:41:52 PDT 2015
LastAccessTime:         UNKNOWN
Protect Mode:           None
Location:               hdfs://192.168.153.129:9000/user/hduser/apple/2015/03/12/09
Partition Parameters:
        COLUMN_STATS_ACCURATE   false
        numFiles                8
        numRows                 -1
        rawDataSize             -1
        totalSize               1215988253
        transient_lastDdlTime   1426974112

# Storage Information
SerDe Library:          com.cloudera.hive.serde.JSONSerDe
InputFormat:            org.apache.hadoop.mapred.TextInputFormat
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:             No
Num Buckets:            -1
Bucket Columns:         []
Sort Columns:           []
Storage Desc Params:
        serialization.format    1
Time taken: 0.305 seconds, Fetched: 43 row(s)
hive (default)>
```

Figure 4.4. External Table Partition

Some Tweet analytics like the active timezones for the generated tweets, retweet analysis are performed on this Table [2] using Hive are shown in Figures 4.5, 4.6. Due to the size constraints configured for the Hadoop cluster daemons, only small portion of the data of around 0.3 GB on partition datehour='2015031208' has been considered for analysis.

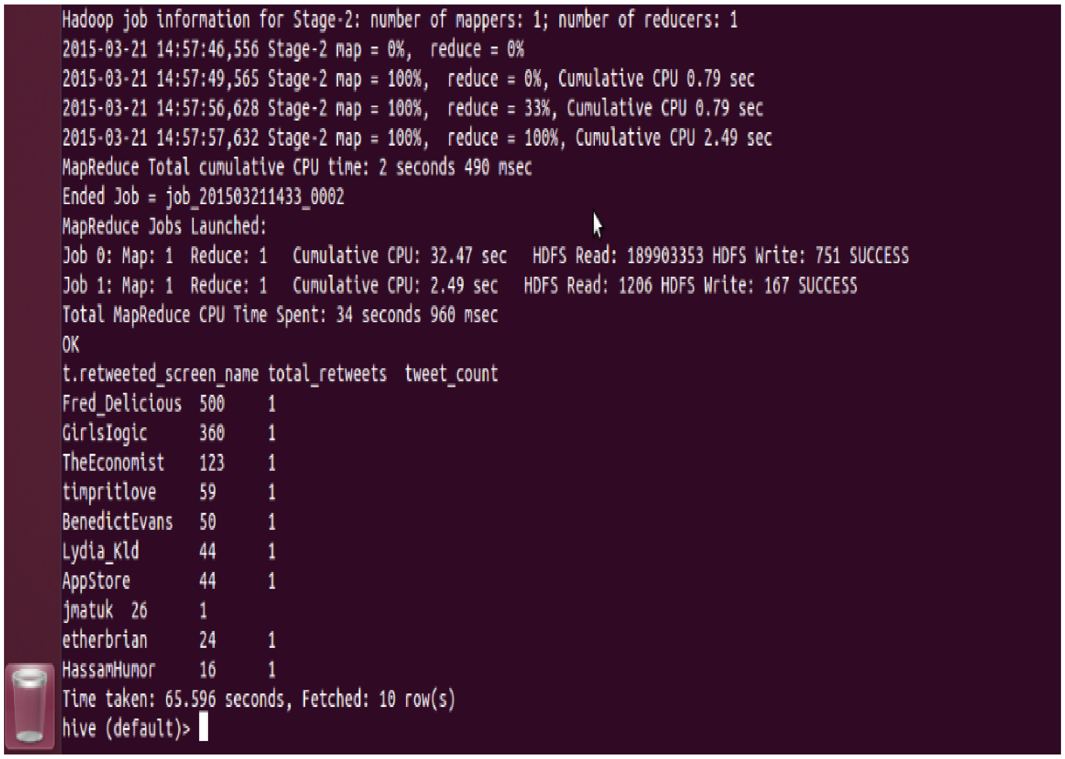


```
  set mapred.reduce.tasks=<number>
Starting Job = job_201503211536_0005, Tracking URL = http://ubuntu:50030/jobdetails.jsp?jobid=job_201503211536_0005
Kill Command = /home/hduser/hadoop-1.2.0/libexec/../bin/hadoop job  -kill job_201503211536_0005
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2015-03-21 16:00:50,525 Stage-2 map = 0%,  reduce = 0%
2015-03-21 16:00:52,531 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.53 sec
2015-03-21 16:01:00,580 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 1.81 sec
MapReduce Total cumulative CPU time: 1 seconds 810 msec
Ended Job = job_201503211536_0005
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 22.46 sec   HDFS Read: 189903353 HDFS Write: 1126 SUCCESS
Job 1: Map: 1  Reduce: 1   Cumulative CPU: 1.81 sec   HDFS Read: 1582 HDFS Write: 320 SUCCESS
Total MapReduce CPU Time Spent: 24 seconds 270 msec
OK
user.time_zone  _c1     total_count
Eastern Time (US & Canada)      Thu     3911
Central Time (US & Canada)      Thu     1739
Pacific Time (US & Canada)      Thu     1305
London  Thu     1303
Mexico City     Thu     1303
Tokyo   Thu     1303
Amsterdam       Thu     1303
Singapore       Thu     870
Atlantic Time (Canada)  Thu     870
Bogota  Thu     870
Alaska  Thu     870
Chennai Thu     870
Caracas Thu     869
Hawaii  Thu     869
Athens  Thu     435
Time taken: 52.257 seconds, Fetched: 15 row(s)
hive (default)>
```

FIGURE 4.5. Analysis of the User Activity on Different Timezones

4.4. Configuring Spark SQL

The Data is now been ingested, cleansed, formatted in Hive and is used for querying to derive some analytics based on the requirements as seen from the above Sections 4.1, 4.3. At a next level of design, Spark SQL [30, 40] which provides sql like queries using Spark is used to access the data on HDFS to compare the same results with Hadoop. Spark SQL builds a schema from the existing RDD called as SchemaRDD [30]. " A SchemaRDD is analogous to a table in a Relational Database" [30]. To use the Spark SQL component, a SQLContext is created by using the SparkContext object in Spark [30]. SQLContext will be a starting point of interaction to use sql like queries. Spark SQL has the ability to derive the schema of the Json data and build a new SchemaRDD [30] . In this usecase as the data ingested is of Json format, a SQLContext is used to infer the Json schema for query analysis. If any intermediate RDD is reused for computation Spark provides a feature called "persist"



```
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2015-03-21 14:57:46,556 Stage-2 map = 0%,  reduce = 0%
2015-03-21 14:57:49,565 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.79 sec
2015-03-21 14:57:56,628 Stage-2 map = 100%,  reduce = 33%, Cumulative CPU 0.79 sec
2015-03-21 14:57:57,632 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.49 sec
MapReduce Total cumulative CPU time: 2 seconds 490 msec
Ended Job = job_201503211433_0002
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 32.47 sec   HDFS Read: 189903353 HDFS Write: 751 SUCCESS
Job 1: Map: 1  Reduce: 1   Cumulative CPU: 2.49 sec   HDFS Read: 1206 HDFS Write: 167 SUCCESS
Total MapReduce CPU Time Spent: 34 seconds 960 msec
OK
t.retweeted_screen_name total_retweets  tweet_count
Fred_Delicious  500     1
GirlsIogic      360     1
TheEconomist    123     1
timpritlove     59      1
BenedictEvans   50      1
Lydia_Kld       44      1
AppStore        44      1
jmatuk  26      1
etherbrian      24      1
HassamHumor     16      1
Time taken: 65.596 seconds, Fetched: 10 row(s)
hive (default)>
```

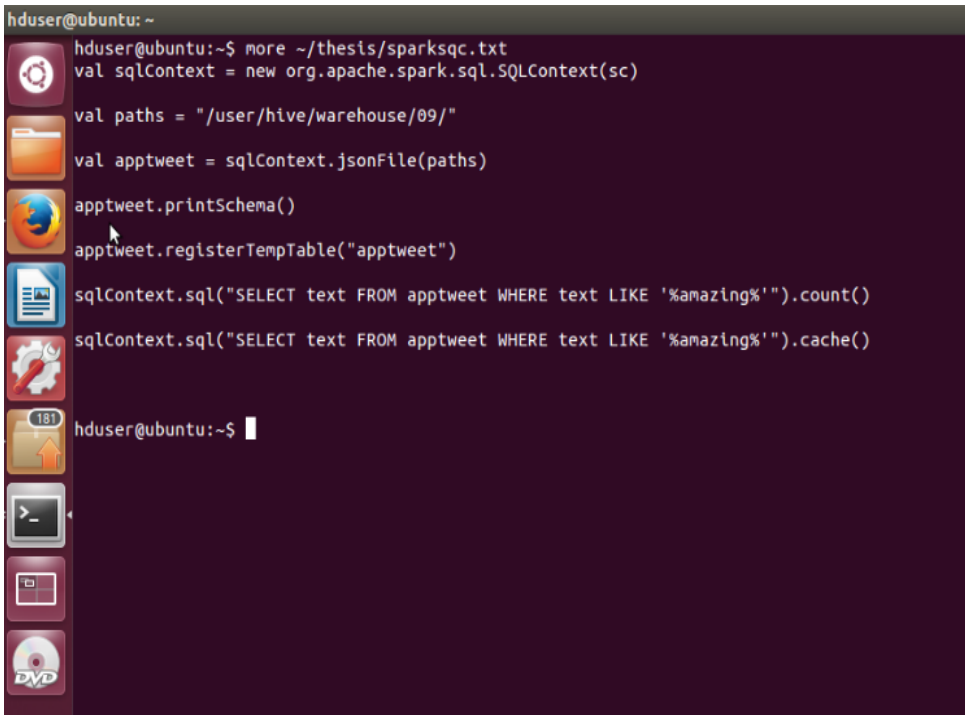
Figure 4.6. Maximum Retweet Analysis

[15] to cache the data. Computing an already cached data can significantly achieve faster results 4.5. The Figure 4.7 is the Spark SQL analysis.

4.5. Results

This section tells about the comparisonal analysis for the queries executed on same datasets both with Hive and Spark. The dataset here is the Hive partition data on date-hour='2015031209' which was discussed earlier in Section 4.3. The counts for the keyword "amazing" on tweets data for the Table apple is analyzed both on Hive as well as Spark.

The Figure 4.8 demonstrates the Hive and Spark SQL query execution times. "Hive.count" is the time taken for the count analysis using Hive. "Spark.count" is the time taken for the count analysis using Spark SQL. "Spark.cache.count" is when the data is cached and performed the count analysis. The first time execution if cache mechanism does not give the better result, but the subsequent count queries can definitely benefit due to caching and can be seen in Figure 4.8.



```
hduser@ubuntu:~$ more ~/thesis/sparksqc.txt
val sqlContext = new org.apache.spark.sql.SQLContext(sc)

val paths = "/user/hive/warehouse/09/"

val apptweet = sqlContext.jsonFile(paths)

apptweet.printSchema()

apptweet.registerTempTable("apptweet")

sqlContext.sql("SELECT text FROM apptweet WHERE text LIKE '%amazing%'").count()

sqlContext.sql("SELECT text FROM apptweet WHERE text LIKE '%amazing%'").cache()


hduser@ubuntu:~$
```

Figure 4.7. Spark SQL JsonRDD Analysis to Analyze the Json format Tweet Data

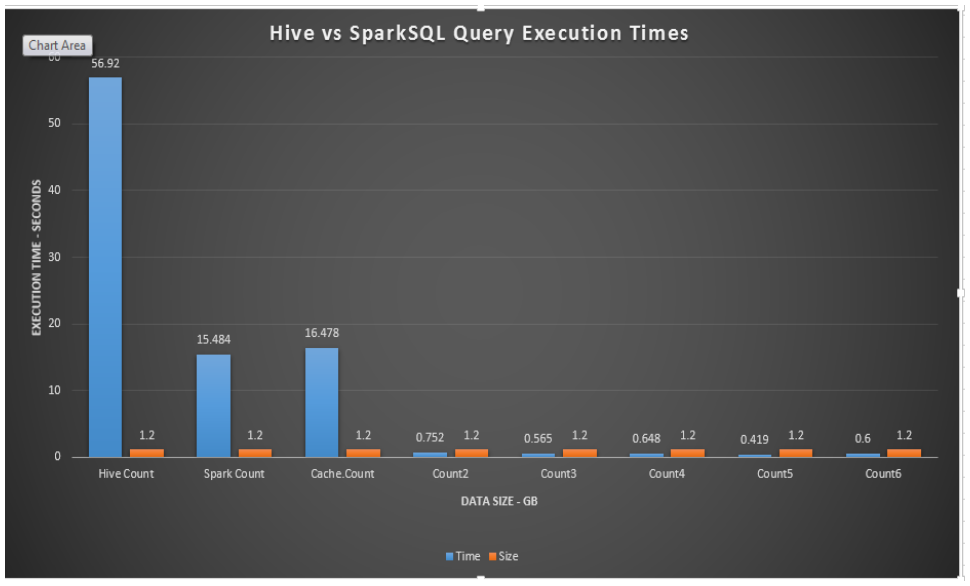


Figure 4.8. Hive vs Spark SQL Query Analysis on the Partition Data

Count Query	Execution Time	Data Size
Hive.count	56.92s	1.2GB
Spark.count	15.484s	1.2GB
Spark.cache.count	16.478s	1.2GB
Spark.count	0.752s	1.2GB
Spark.count	0.565s	1.2GB
Spark.count	0.648s	1.2GB
Spark.count	0.419s	1.2GB
Spark.count	0.60s	1.2GB

TABLE 4.1. Hive vs Spark SQL Query Analysis.

4.6. Ganglia Metrics

During the query execution analysis using Hive, some cluster metrics have been analyzed on the CPU, Memory, Process loads. The overall Memory of the cluster is 12GB 4.1, cores are 6 4.1, nodes are 3.

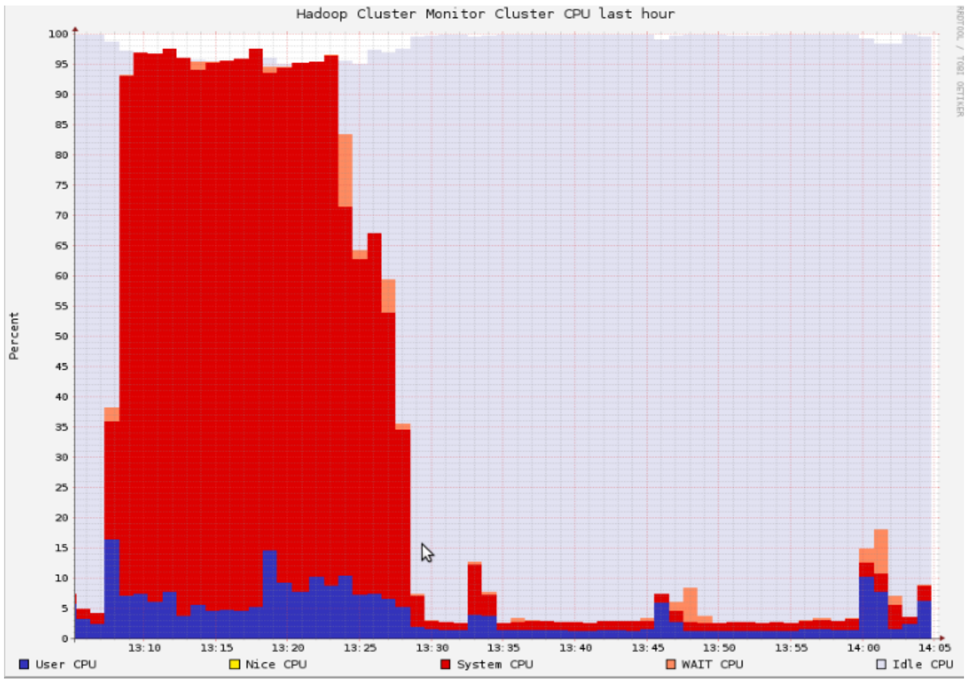


FIGURE 4.9. Ganglia CPU Cluster Metrics of Hadoop Cluster

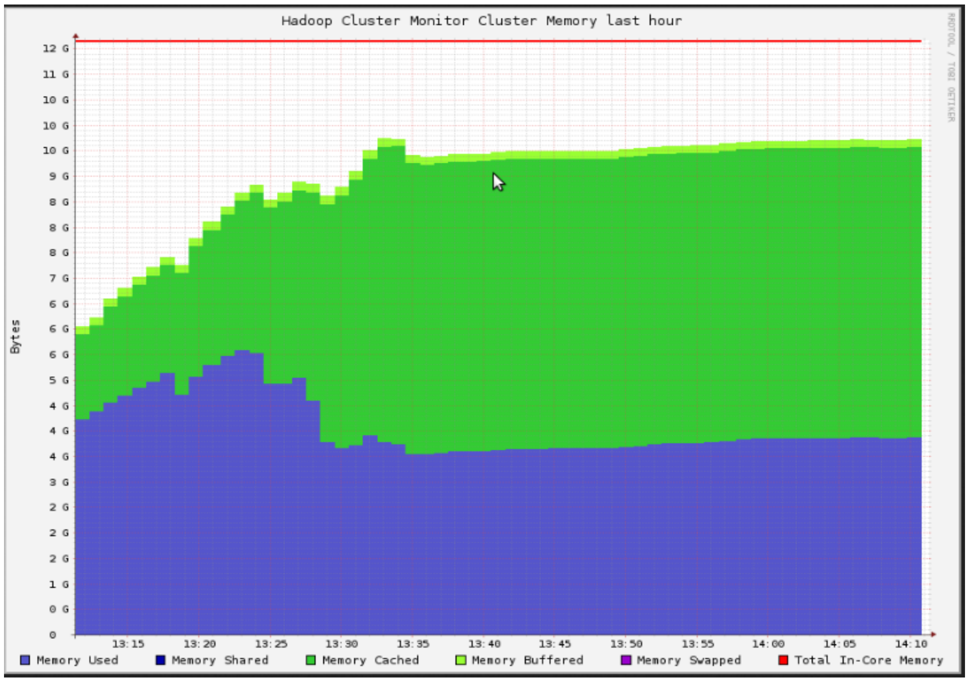


Figure 4.10. Ganglia MEMORY Cluster Metrics of Hadoop Cluster

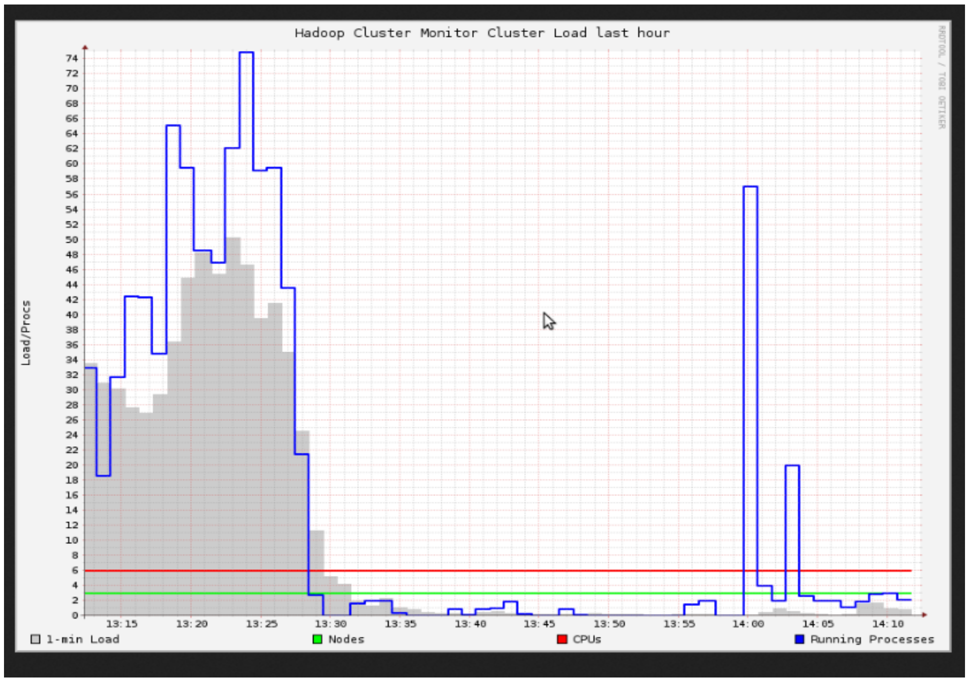


Figure 4.11. Ganglia PROCESS Load Metrics of Hadoop Cluster

CHAPTER 5

CONCLUSIONS AND FUTUREWORK

This thesis presents an Open Data Architecture in bulding a Data flow lake by using the available open source distributed technologies on a low commodity hardware. This approach uses Hadoop Distributed File System as a storage for reliability and MapReduce as the main programming model for the computation.

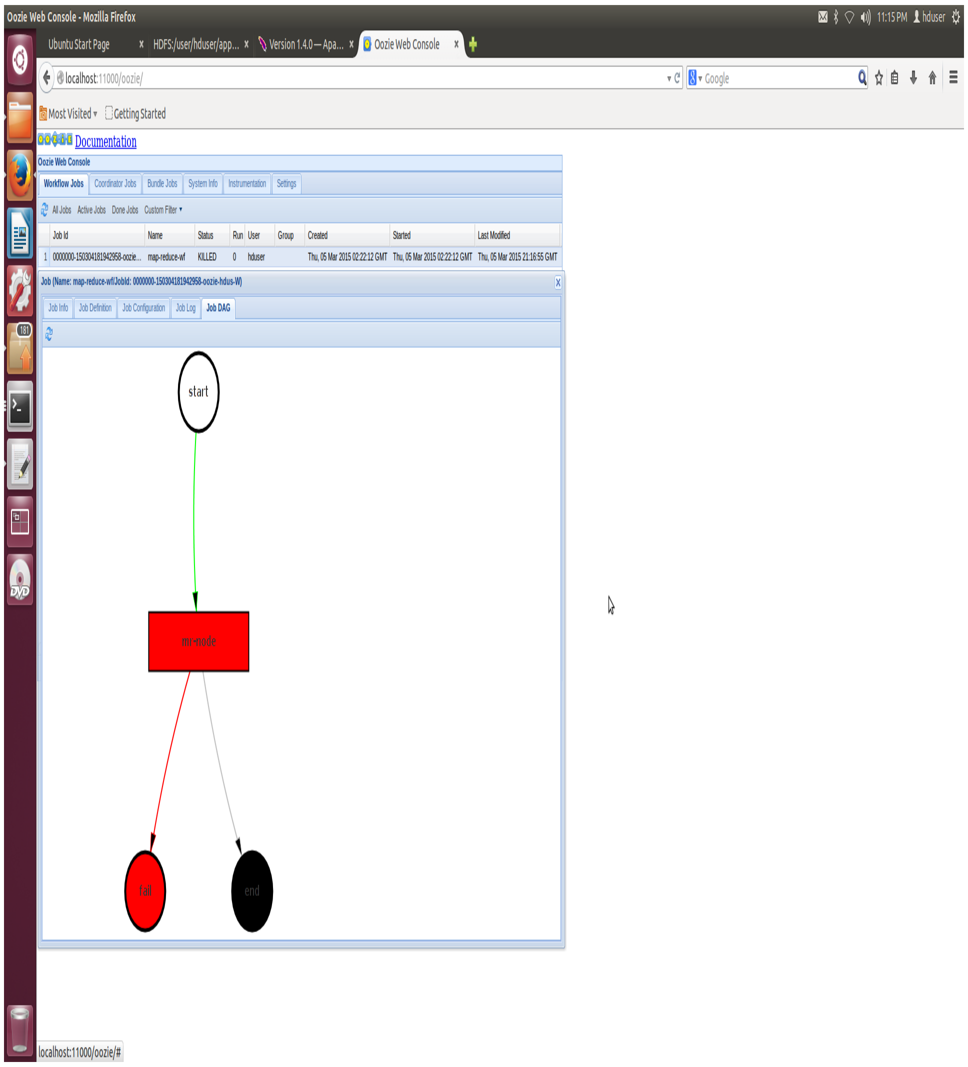
This study also analyzes the uses of frameworks like Flume and Hive in order to work on the semi structured data formats. This study reproduces the data flow architecture by using the existing ones depending on the usecase requirement on a low cost hardware.

Finally, this thesis also discusses the integration of distributed framework components at a single level of design to improve execution time and for better managability.

5.1. Future Work

The data ingested for analysis for this research is limited to a small size due to Hardware constraints. In this study, Flume is configured to bring the data and store it in a hourly fashion on HDFS 4.1. The Partitions in this case study is limited to 3 since the data ingested is only limited to 3 hours of data using Flume agent. At higher level of Production Environments when Flume is used to collect large amounts of data in hourly fashion, there is a need to add more Partitions in Hive to look at the latest view of the Data for analytics. The data to an External table can be made available by adding Partitions as discussed in section 4.3. As the data increases, creating the Partitions manually in Hive is a bit labourious and not a good technique. This process of adding partitions can be scheduled through a scheduler. Apache Oozie$^{\text{TM}}$"is a scheduling framework to manage and automate Hadoop jobs" [23, 18]. These collection of jobs are called as actions and they form a "control dependency" [23] DAGs. "These actions are referred as a Directed Acyclic Graph jobs because, the later job cannot run until unless the previous action is completed" [23].

The Figure 5.1 is a sample oozie job flow DAG. Oozie workflows are controlled by action nodes and control flow nodes. Oozie currently supports MapReduce, Pig, Hive, Sqoop,



	Job Id	Name	Status	Run	User	Group	Created	Started	Last Modified
1	0000000-150304181942958-oozie...	map-reduce-wf	KILLED	0	hduser		Thu, 05 Mar 2015 02:22:12 GMT	Thu, 05 Mar 2015 02:22:12 GMT	Thu, 05 Mar 2015 21:16:55 GMT

Job (Name: map-reduce-wf/JobId: 0000000-150304181942958-oozie-hdus-W)

FIGURE 5.1. Oozie Job

Java and Shell script workflow jobs [23]. Oozie provides Coordinator jobs that can be configured to trigger actions at regular intervals of time. Since the data ingesting through Flume is in a hourly fashion and for Hive to have the latest view of the data, the "ALTER TABLE" to add partitions can be configured in the Oozie Coordinator job to run at every 60 minutes. This mechanism can significantly automate the process to add hourly Partition data onto Hive which can be seamlessly used to derive analytics on the latest set of Data.

BIBLIOGRAPHY

[1] Edward Capriolo, Dean Wampler, and Jason Rutherglen, *Programming hive*, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., October 2012.

[2] Natkins Jon cloudera BLOG, *http://blog.cloudera.com/blog/2012/09/analyzing-twitter-data-with-hadoop/*.

[3] cloudera github, *https://github.com/cloudera/cdh-twitter-example*.

[4] Jeffrey Dean and Sanjay Ghemawat, *Mapreduce: Simplified data processing on large clusters*, Commun. ACM 51 (2008), no. 1, 107–113.

[5] Apache Hive DML, *https://cwiki.apache.org/confluence/display/hive/languagemanual+dml*.

[6] Apache Flume, *http://flume.apache.org/releases/content/1.4.0/flumeuserguide.html*.

[7] Ganglia, *http://ganglia.sourceforge.net/*.

[8] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, *The google file system*, Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (New York, NY, USA), SOSP '03, ACM, 2003, pp. 29–43.

[9] Ganglia github, *https://github.com/ganglia/monitor-core/wiki/ganglia-quick-start*.

[10] Apache Hive Developer Guide, *https://cwiki.apache.org/confluence/display/hive/developerguide*.

[11] Apache Hadoop, *Apache hadoop*, 2011.

[12] Apache HDFS, *https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html*.

[13] Apache Hive, *https://hive.apache.org/*.

[14] Steve Hoffman, *Apache flume: Distributed log collection for hadoop*, Packt Publishing Ltd, 2013.

[15] Karau Holden, Andy Konwinski, Patrick Wendell, and Zaharia Matei, *Learning spark*, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., February 2015.

[16] Alex Holmes, *Hadoop in practice*, Manning Publications Co., 2012.

[17] Hortonworks, *http://hortonworks.com/hadoop/flume/*.

[18] Mohammad Islam, Angelo K. Huang, Mohamed Battisha, Michelle Chiang, Santhosh

Srinivasan, Craig Peters, Andreas Neumann, and Alejandro Abdelnur, *Oozie: Towards a scalable workflow management system for hadoop*, Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies (New York, NY, USA), SWEET '12, ACM, 2012, pp. 4:1–4:10.

[19] Natkins Jon, *http://blog.cloudera.com/blog/2012/10/analyzing-twitter-data-with-hadoop-part-2-gathering-data-with-flume/.*

[20] Rohit Khare, Doug Cutting, Kragen Sitaker, and Adam Rifkin, *Nutch: A flexible and scalable open-source web search engine*, Oregon State University 32 (2004).

[21] Vania Marangozova-Martin and Vania Marangozova, *Introduction to distributed systems.*

[22] Thomas Nägele and FW Vaandrager, *Mapreduce framework performance comparison*, (2013).

[23] Apache Oozie, *http://oozie.apache.org/.*

[24] Pivotal, *http://blog.pivotal.io/big-data-pivotal/news-2/new-federation-business-data-lake-should-be-your-silver-bullet-for-big-data-success.*

[25] PostgreSQL, *http://www.postgresql.org/.*

[26] Philip Russom et al., *Big data analytics*, TDWI Best Practices Report, Fourth Quarter (2011).

[27] Lee Scott, *https://www.digitalocean.com/community/tutorials/introduction-to-ganglia-on-ubuntu-14-04.*

[28] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler, *The hadoop distributed file system*, Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (Washington, DC, USA), MSST '10, IEEE Computer Society, 2010, pp. 1–10.

[29] Apache Spark, *http://spark.apache.org/.*

[30] Apache Spark SQL, *http://spark.apache.org/docs/1.2.1/sql-programming-guide.html.*

[31] Apache Sqoop, *http://sqoop.apache.org/.*

[32] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh

Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy, *Hive: A warehousing solution over a map-reduce framework*, Proc. VLDB Endow. 2 (2009), no. 2, 1626–1629.

[33] Kathleen Ting and Jarek Jarcec Cecho, *Apache sqoop cookbook*, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., July 2013.

[34] Scaling Hadoop to 4000 nodes at Yahoo, *https://developer.yahoo.com/blogs/hadoop/scaling-hadoop-4000-nodes-yahoo-410.html*.

[35] Ubuntu, *http://www.ubuntu.com/*.

[36] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler, *Apache hadoop yarn: Yet another resource negotiator*, Proceedings of the 4th Annual Symposium on Cloud Computing (New York, NY, USA), SOCC '13, ACM, 2013, pp. 5:1–5:16.

[37] vmware, *http://www.vmware.com/*.

[38] Tom White, *Hadoop: The definitive guide*, 3rd ed., O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., May 2012.

[39] title=http://en.wikipedia.org/wiki/Fault_tolerance Wikipedia.

[40] Reynold S Xin, Josh Rosen, Matei Zaharia, Michael J Franklin, Scott Shenker, and Ion Stoica, *Shark: Sql and rich analytics at scale*, Proceedings of the 2013 ACM SIGMOD International Conference on Management of data, ACM, 2013, pp. 13–24.

[41] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica, *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing*, Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, USENIX Association, 2012, pp. 2–2.

[42] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica, *Spark: cluster computing with working sets*, Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, 2010, pp. 10–10.