Resource Comparison between Hadoop and Spark Architecture by Performing Data Analysis

Mukesh Sahu
Computer and Information Sciences
University of Michigan
Dearborn MI US
mukeshs@umich.edu

Jaya Shree Jilkara
Computer and Information Science
University of Michigan
Dearborn MI US
jshree@umich.edu

Seema Sharanappa Kanaje
Computer and Information Science
University of Michigan
Dearborn MI US
skanaje@umich.edu

***Abstract*— Today we are surrounded by a massive amount of unstructured data mostly due to advancement of technology and social media. The main objective of this paper is to build a Big data processing infrastructure for analyzing data using HDFS and Spark-based NoSQL systems. We aim to perform analysis on resource allocation and NoSQL. We aim to obtain performance results by deploying the dataset on Hadoop and Spark**

Keywords—Hadoop MapReduce, Spark, R, RStudio, Data Analysis, Word Cloud, Resource, CPU, Memory

I. Introduction

According to IBM recent data we are surrounded with a huge amount of data. As per them 90% of data was produced in the last two years. These data are results of social media, ecommerce sites, various applications, and multiple weblinks which were only possible because of the development of a new database called NoSQL. Many of the data scientists today are using the open-source statistical modelling language R when working with massive data and Hadoop. R is the most favorite tool of data scientists because of its wide variety of functions that includes data manipulation and statistical modelling. As our project involves significant amounts of data for analyzing and visualization, we made use of R with Hadoop. We implemented this with the help of novel packages such as RHadoop, RMR, RHIPE etc.

Therefore, the idea is to build a Big Data processing infrastructure for analyzing data using HDFS and Spark based NoSQL systems. This will require us to obtain unstructured raw data from social media sites and store them on Hadoop Distributed File System and Spark system to process this data analytic using MapReduce and SVM algorithm using RStudio. Further, we will be analyzing the resource utilization for both the architecture system [1][2].

II. Background/Motivation

Dataset and architecture storing the dataset are the main factors that decide which classifier or algorithm would be suited the most to perform certain analysis. We need to select the classifier based on the type of dataset and architecture, available time duration and resources we are having because sometimes one classifier does better on one architecture than the other due to dependency and other factors of the data in the dataset. We aim to obtain performance results from the deploying the dataset on Hadoop and spark system [3].

This implementation will help us understand the MapReduce operation on Spark as well as Hadoop and will make us familiar with these infrastructures. Moreover, it will assist us with resource allocation understanding of the system and NoSQL.

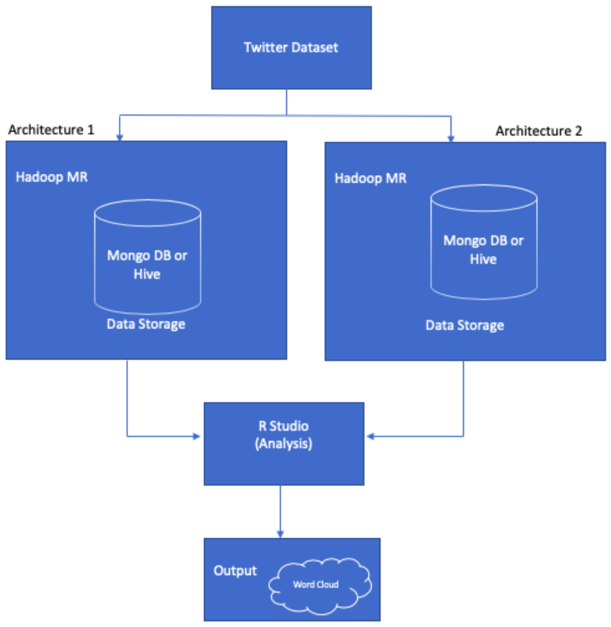


Fig 1. Project Flow Chart for Basic Understanding

A. *Application Programming Interface*

You can communicate from one application to another one with the help of an API. Consider a scenario wherein you wanted to access the functionality of another application, for example: Consider our scenario where we wanted to access all the tweets with #COVID. The easiest way to do this would be for Twitter to provide a way to query their application to retrieve that data.

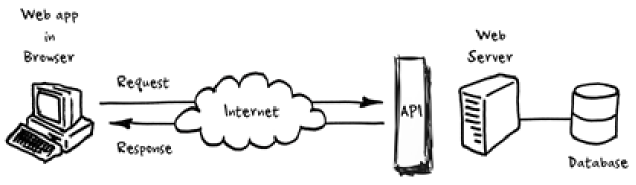
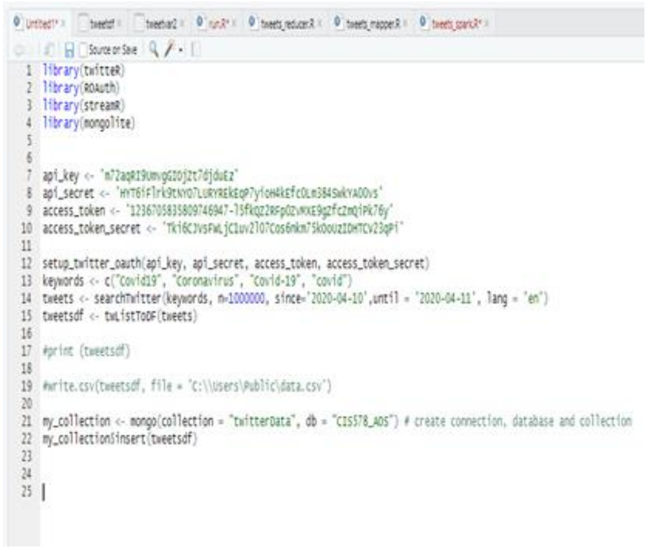


Fig 2. Application programming interface

Some of the APIs that are supported by Twitter are Rest API and Stream API. In the Rest API, a request is sent to the client by server which then sends its Response. Whereas in stream API, the server itself updates the client on any changes.

B. *Stream API*

For this project, we are using Stream API for obvious reasons to get the information from Twitter.

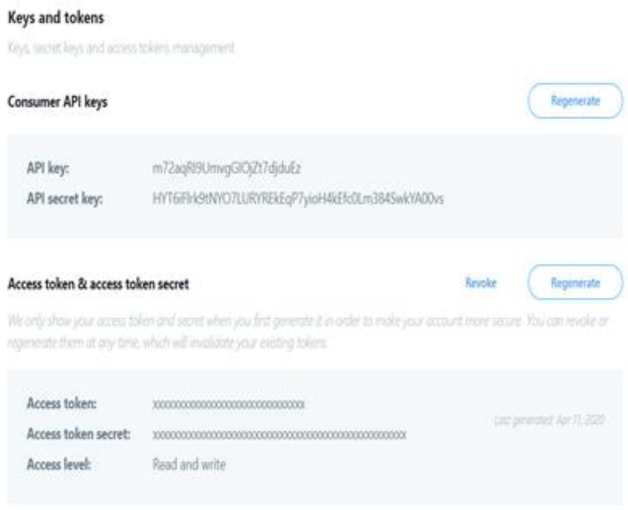


```r
library(twitteR)
library(ROAuth)
library(streamR)
library(mongolite)


api_key <- 'm72aqRI9UmvgGIOjZt7djduEz'
api_secret <- 'HYT6iFlrk9tNYO7LURYREkEqP7yioH4kEfcOLm384SwkYA0Ovs'
access_token <- '1236705835809746947-15fkQZ2RFpOzVmXE9gzfcZmQiPk76y'
access_token_secret <- 'Tki6CJVsFWLjC1uv2l07Cos6nkm75kOOUZIDHTCV23qPi'

setup_twitter_oauth(api_key, api_secret, access_token, access_token_secret)
keywords <- c("Covid19", "Coronavirus", "Covid-19", "covid")
tweets <- searchTwitter(keywords, n=1000000, since='2020-04-10',until = '2020-04-11', lang = 'en')
tweetsdf <- twListToDF(tweets)

#print (tweetsdf)

#write.csv(tweetsdf, file = 'C:\\Users\Public\data.csv')

my_collection <- mongo(collection = "twitterData", db = "CIS578_ADS") # create connection, database and collection
my_collection$insert(tweetsdf)


|
```

Fig 3. Code for Stream Api

In the above code, you can see that we have passed the token for the setup. For the keywords, we have used- ‘Covid’ 19’, ‘Coronavirus’, ‘Covid - 19’ and ‘Covid’ to retrieve all the tweets related to Covid. It also contains a variable with the name ‘tweet’ which stores nearly million data. As Twitter can store data up to only seven days, we were successful in retrieving data from 5th April to 12th April’ 20.

Here we have used libraries like ‘twitterR’ to connect with Twitter Service using API key and access tokens. ‘R0Auth’ library in ‘R’ was used for passing these credential parameters. RStudio has a functionality to convert all the data in its own readable data frame, therefore, we passed ‘tweetsdf’ parameter to display data obtained from twitter in ‘R’ data frame. These data were then analyzed and passed in MongoDB using the variable name ‘my_collection’. We made use of insert statement to insert all the JSON data obtained from Twitter.



Keys and tokens

Keys, secret keys and access tokens management

Consumer API keys

API key: m72aqRI9UmvgGIOjZt7djduEz
API secret key: HYT6iFlrk9tNYO7LURYREkEqP7yioH4kEfc0Lm384SwkYA0Ovs

Access token & access token secret

We only show your access token and secret when you first generate it in order to make your account more secure. You can revoke or regenerate them at any time, which will invalidate your existing tokens.

Access token: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Access token secret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Access level: Read and write

Last generated: Apr 11, 2020

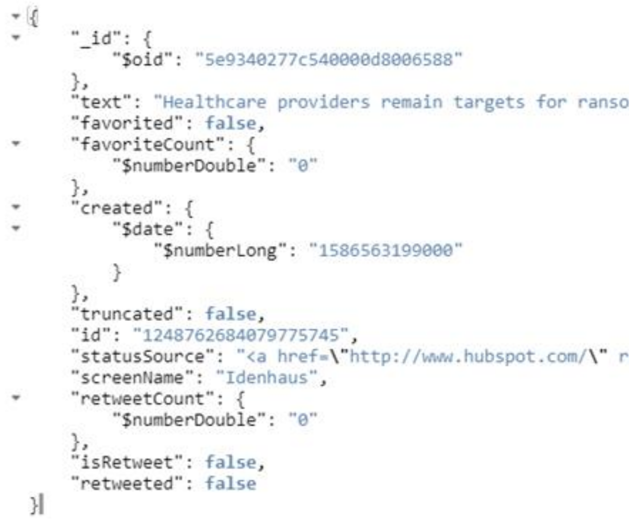
Fig 4. Twitter Application Developer Mode Interface

For this, we had to request the access token by creating an application on Twitter’s developer account. As per their policy, the user is required to create a developer account by putting all the details of what the application will be doing and how exactly we will be using their data. Once we have all these information, we have to pass these in script written in ‘R’ after importing all the necessary libraries.

C. *Data Analysis*

To perform data analysis, we will be analyzing the data stored in NoSQL dataset on RStudio by both the Hadoop MapReduce and Spark architectures by calculating the frequency of the word and forming the ‘Word Cloud’. The resources like CPU, Memory, and I/O is used as the performance metrics.

For analysis, we are considering about 1M tweets from twitter users. The raw data from twitter is in the JSON format which doesn’t require any preprocessing or modification to data.



```
{
    "_id": {
        "$oid": "5e9340277c540000d8006588"
    },
    "text": "Healthcare providers remain targets for ranso
    "favorited": false,
    "favoriteCount": {
        "$numberDouble": "0"
    },
    "created": {
        "$date": {
            "$numberLong": "1586563199000"
        }
    },
    "truncated": false,
    "id": "1248762684079775745",
    "statusSource": "<a href=\"http://www.hubspot.com/\" r
    "screenName": "Idenhaus",
    "retweetCount": {
        "$numberDouble": "0"
    },
    "isRetweet": false,
    "retweeted": false
}
```

Fig 5. Data Sample Obtained from Twitter

Figure 5, is the representation of the data or the tweets obtained from Twitter in JSON format. Here, ‘text’ is the actual tweet posted by the user, ‘favorite count’ is number of likes that tweet has obtained, ‘screenname’ is Username of the account using which this was tweeted, ‘retweet Count’ is the number of times the data has been shared and ‘created’ is tweet created date.

D. *Databases*

Here, since we knew that we need to gather a large amount of data, approximately 1 million data and these data would be unstructured. Therefore, we would be required to select a perfect database that can be accessed by ‘RStudio’ and be reliable. Hive, Cassandra, and MongoDB were few of the option that we were exploring.

1) *Hive*

It is an open source data warehouse that is built on top of Hadoop, which is used for structured as well as semi structured data that provides data summarization, analysis and query. It stores schema in a database. Processing of data is done by using HDFS. The query language used by Hive automatically translates queries into MapReduce jobs. However, it does not support transaction processing that involves high write operations. In-spite of being fast, scalable and highly extensible, it does not offer real-time queries.

2) *Cassandra*

It is an open source and free wide column-oriented database management system introduced by Apache, which is used by many companies such as Facebook, Netflix. It is designed to handle large amounts of data providing high availability. It follows master-slave architecture. It works on peer-to-peer architecture, using which it overcomes the problem of scalability and high availability. In this, each node is assigned with a role. Even if node failures occur, the data is distributed automatically across all the nodes. Cassandra, however, does not have a map-reduce of its own. Therefore, it integrates with Hadoop.

3) *MongoDB*

It is an open source document-oriented database management system which is designed to handle applications that use structured and unstructured data. In this, the data is stored in documents and not in tabular format. MongoDB also has relational database features such as indexing, queries, replication. It also has map-reduce of its own for the distributed processing of the data. However, to use this, the program must be written in JavaScript which limits the scope for data processing as not many data processing libraries are written in JavaScript.

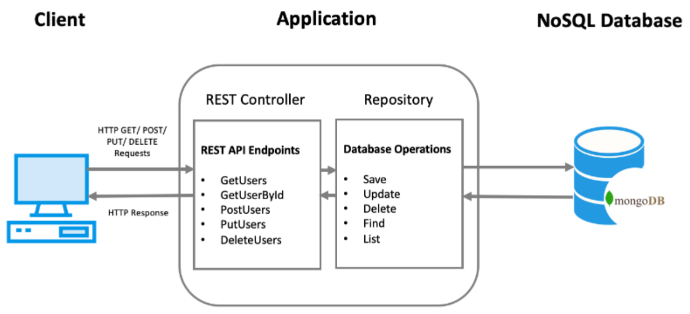
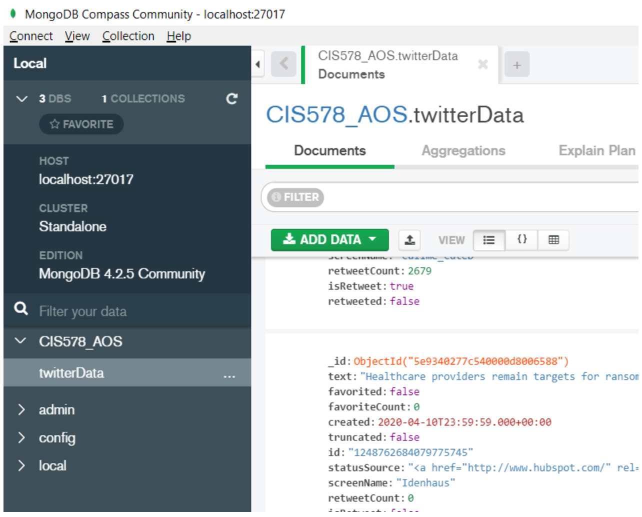


Fig 6. MongoDB API functionality

As MongoDB is one of the reliable and easily accessible using HTTP POST/GET method. We decided to use that for our project implementation. After pulling the tweets from RStudio, Streaming Twitter API and loaded them onto MongoDB. We pulled around 1 million tweets which was about 7days.



```
retweetCount: 2679
isRetweet: true
retweeted: false

_id: ObjectId("5e9340277c540000d8006588")
text: "Healthcare providers remain targets for ransom
favorited: false
favoriteCount: 0
created: 2020-04-10T23:59:59.000+00:00
truncated: false
id: "1248762684079775745"
statusSource: "<a href="http://www.hubspot.com/" rel=
screenName: "Idenhaus"
retweetCount: 0
```

Fig 7. Document Database in MongoDB

We created a database instance name ‘CIS578_AOS’ with collection named ‘twitterData’ running on port 27017 on the local host.

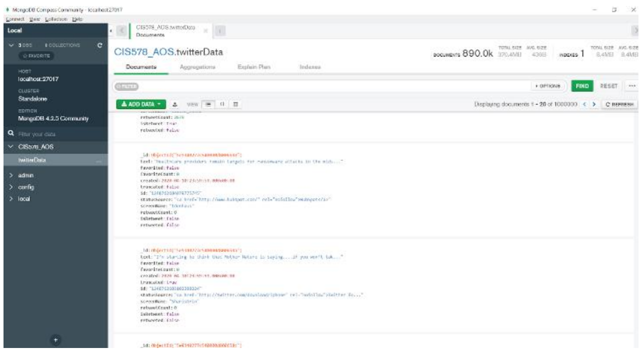


Fig 8. MongoDB Interface

Each tweet looked as displayed in the below screenshot. We looked for the tweets 'COVID' hashtags in subjects. From Hadoop and Spark, we queried data from MongoDB to get the count of each word.

E. *Tools and Libraries*

Here, we used the data analyses using R/RStudio

- Library - rmr2 and rhdfs

These libraries are used to connect RStudio with HDFS and use MapReduce.

- Library - streamR, ROAuth, and twitteR

We have used these libraries on RStudio to obtain or gather data from Twitter.

- Library - DBI, dplyr, and odbc:

We have stored the data in MongoDB and load that up on Hadoop and use the database using R studio by implementing R packages such as DBI, dplyr, and odbc [4][5].

F. *Algorithm*

To create a word cloud there are several packages to be installed. For example, 'RColorBrewer' is a package for Colors, 'tm' is for text mining. The first step to generating the word cloud is installing these libraries.

The next step is cleaning the data. In this step, special characters need to be removed from the data and numbers, punctuation, white spaces are also to be removed. There are certain common words, like an, the, etc. Such common words are also removed so that these words don't appear in the word cloud.

One of the packages installed is 'tm' package which is used for text mining and one of the functions of 'tm' package called 'tm_map' is used to remove all this unnecessary data. After cleaning the data, the third step is to create a document term matrix. This matrix is a data frame with one column for all words and the other column for the frequency of each word. The functions for generating this matrix come under the 'tm' package. And then the last step is creating the word cloud and for this the 'word cloud' package is required. And using this package and function 'word cloud' under it by passing the right parameters, a word cloud is created.

III. METHODOLOGY

A. *Spark*

It is a distributed data processing engine with a cluster computing framework. It uses both persistent as well as Resilient Distributed Datasets. Spark does not have a distributed file system like Hadoop, but it is compatible to use Hadoop file systems. In case of spark, it does not need to spend time moving data to and from the disk, which makes it faster as it does everything in memory. Spark is also known for real time stream processing [6].

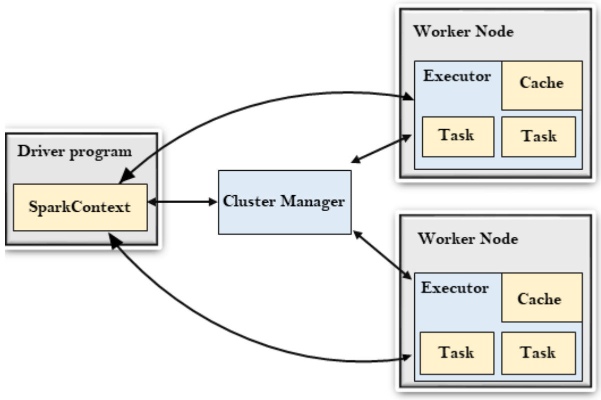


Fig 9. Spark Architecture

It is a distributed data processing engine with a cluster computing framework. It uses both persistent as well as Resilient Distributed Datasets.

Spark does not have a distributed file system like Hadoop, but it is compatible to use Hadoop file systems. In case of spark, it does not need to spend time moving data to and from the disk, which makes it faster as it does everything in memory. Spark is also known for real time stream processing.

B. *Hadoop MapReduce*

As we are working on an HDFS system trying to analyze the frequency of words to form word clouds, we have decided to use MapReduce. It is a processing technique for large problems or big dataset. It contains two tasks Map and Reduce, map is responsible for taking a set of data and converting it into another set and reduce is responsible for taking the data from map as input and converting it into key-value pairs [5].

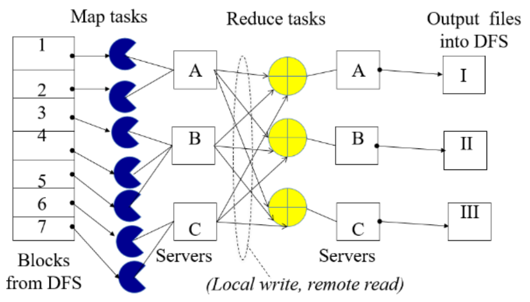


Fig 10. MapReduce Pictorial Implementation [7]

Hadoop is basically a cluster that consists of components connected over a dedicated network to function as a central data processor. There are many features that make Hadoop a good solution like fault-tolerance, high availability, data locality, scalability, distributed storage, replication, etc. It is important to ponder that batch processing is efficiently performed in Hadoop than Spark [5][6].

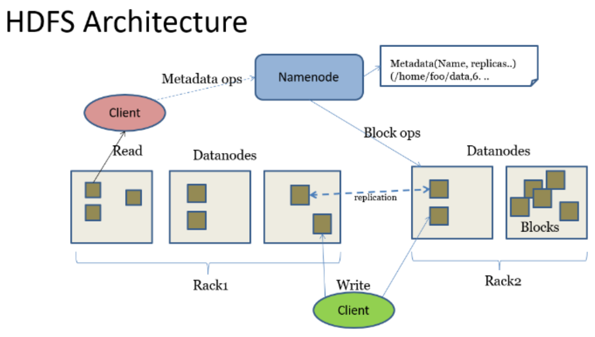


Fig 11: HDFS Architecture [7]

We can use Hadoop as a storage. The good thing about HDFS is that it provides the distributed storage of the massive datasets altogether along with high availability. This implementation requires no extra software to be installed. The best thing is that the data loss is being handled by HDFS itself.

C. *WordCloud*

This is one of the form or way of doing data analysis by making algorithm on architecture to count the number of word and store its prominence.

We used word clouds to show greater prominence which appeared often in tweet data. Word Cloud is also known as text clouds or tag clouds. It is basically the collection of words in different sizes depending upon the frequency of the word.

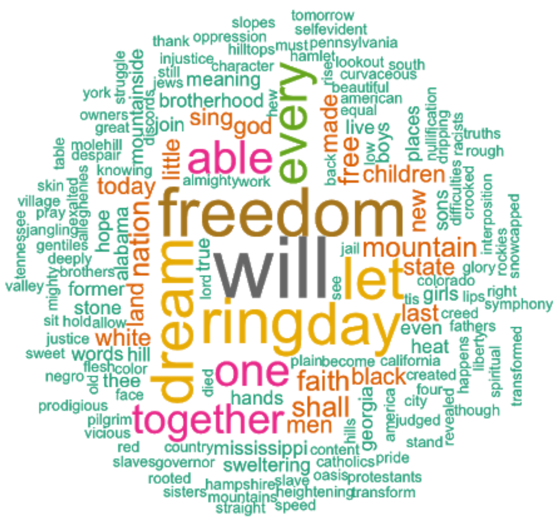


Fig 12: Word Cloud Sample

In figure 12, we see 'WILL' is large and bold which reflects that the maximum number of times appeared in text source.

We are taking Twitter dataset containing several instances of tweets. After which we are processing the data removing special characters and spaces. Tokenize the collection of tweets into vocabulary words and create the vector representation of words presence with its frequency. Briefly, convert the text document into a matrix of words count. These data collection from twitter is done using R and a CSV file will be generated, this is further converted to JSON file format.

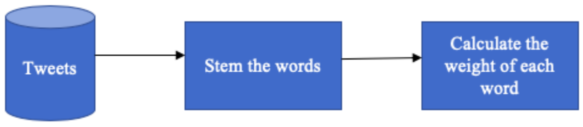


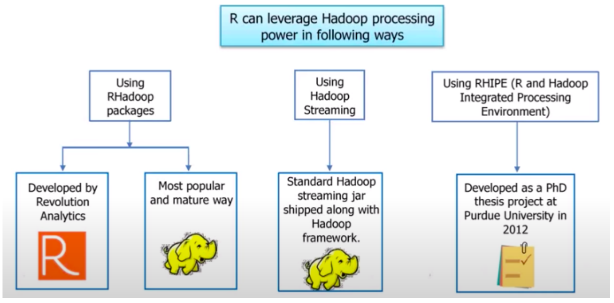
Fig 13: Methodology

IV. Implementation

A. *Hadoop MapReduce with 'R'*

Hadoop allows the processing of large amounts of data parallelly. Using R with Hadoop, we can utilize all the benefits of Hadoop. Moreover, we can get horizontal scalability of statistical calculations by using R with Hadoop.

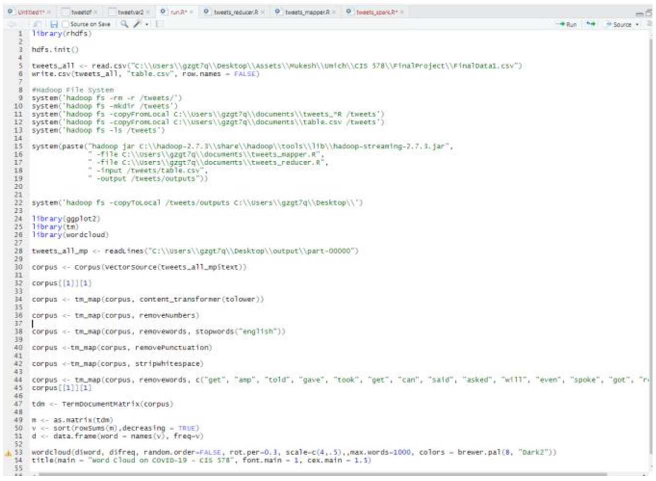
We can integrate R with Hadoop either by using RHadoop packages or by using Hadoop streaming or by using RHIPE (R and Hadoop Integrated Processing Environment). Hadoop Streaming provides you with an advantage that there is no restriction on the language used for Mappers and Reducers.



R can leverage Hadoop processing power in following ways
Using RHadoop packages
Using Hadoop Streaming
Using RHIPE (R and Hadoop Integrated Processing Environment)
Developed by Revolution Analytics
Most popular and mature way
Standard Hadoop streaming jar shipped along with Hadoop framework.
Developed as a PhD thesis project at Purdue University in 2012

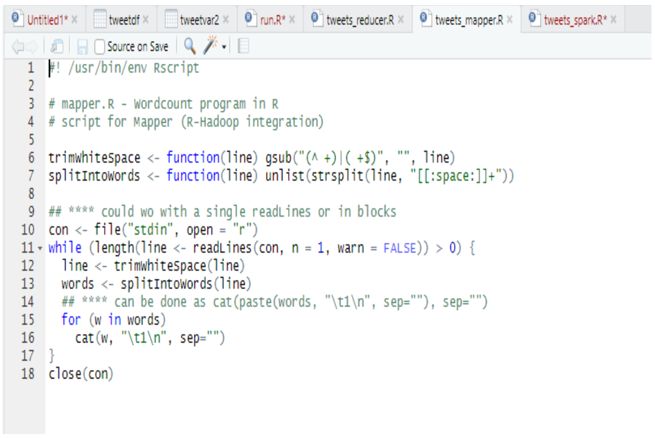
Fig 13. Hadoop integration with R

RHIPE is the first known method used for integration purpose, which is developed by students at Purdue University. In our project, we made use of RHadoop packages which is developed by Revolution Analytics and is currently the most popular way of integration. The RHadoop packages include libraries such as RMR, RHBASE and RHDFS. R-MapReduce (RMR) is basically an interface for running MapReduce jobs using R. On the other hand, RHBASE is for reading, writing and manipulating the data with HBASE, RHDFS is an interface to HDFS which means we can interact with HDFS using R.



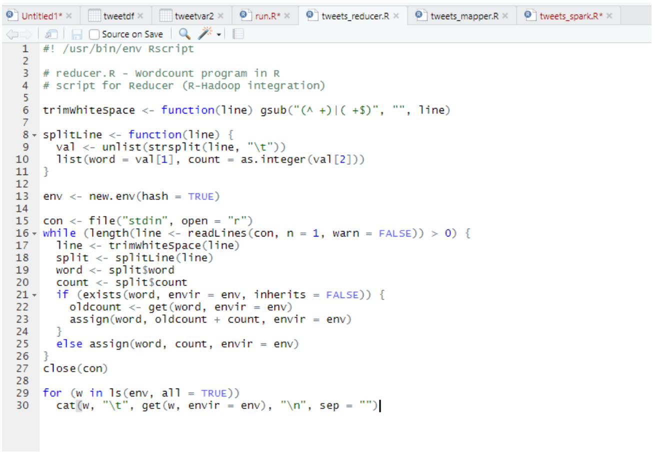
```
library(rhdfs)

hdfs.init()

tweets_all <- read.csv("C:\\Users\\gzgt7q\\Desktop\\Assets\\Mukesh\\Umich\\CIS 578\\FinalProject\\FinalData1.csv")
write.csv(tweets_all, "table.csv", row.names = FALSE)

#Hadoop File System
system('hadoop fs -rm -r /tweets/')
system('hadoop fs -mkdir /tweets')
system('hadoop fs -copyFromLocal C:\\Users\\gzgt7q\\documents\\tweets_*R /tweets')
system('hadoop fs -copyFromLocal C:\\Users\\gzgt7q\\documents\\table.csv /tweets')
system('hadoop fs -ls /tweets')

system(paste("hadoop jar C:\\hadoop-2.7.3\\share\\hadoop\\tools\\lib\\hadoop-streaming-2.7.3.jar",
             " -file C:\\Users\\gzgt7q\\documents\\tweets_mapper.R",
             " -file C:\\Users\\gzgt7q\\documents\\tweets_reducer.R",
             " -input /tweets/table.csv",
             " -output /tweets/outputs"))


system('hadoop fs -copyToLocal /tweets/outputs C:\\Users\\gzgt7q\\Desktop\\')

library(ggplot2)
library(tm)
library(wordcloud)

tweets_all_mp <- readLines("C:\\Users\\gzgt7q\\Desktop\\output\\part-00000")

corpus <- Corpus(VectorSource(tweets_all_mp$text))

corpus[[1]][1]

corpus <- tm_map(corpus, content_transformer(tolower))

corpus <- tm_map(corpus, removeNumbers)

corpus <- tm_map(corpus, removeWords, stopwords("english"))

corpus <-tm_map(corpus, removePunctuation)

corpus <-tm_map(corpus, stripWhitespace)

corpus <- tm_map(corpus, removeWords, c("get", "amp", "told", "gave", "took", "get", "can", "said", "asked", "will", "even", "spoke", "got", "r
corpus[[1]][1]

tdm <- TermDocumentMatrix(corpus)

m <- as.matrix(tdm)
v <- sort(rowSums(m),decreasing = TRUE)
d <- data.frame(word = names(v), freq=v)

wordcloud(d$word, d$freq, random.order=FALSE, rot.per=0.3, scale=c(4,.5),,max.words=1000, colors = brewer.pal(8, "Dark2"))
title(main = "Word Cloud on COVID-19 - CIS 578", font.main = 1, cex.main = 1.5)

```

Fig 14. Hadoop MapReduce in R



```
#! /usr/bin/env Rscript

# mapper.R - Wordcount program in R
# script for Mapper (R-Hadoop integration)

trimWhiteSpace <- function(line) gsub("(^ +)|( +$)", "", line)
splitIntoWords <- function(line) unlist(strsplit(line, "[[:space:]]+"))

## **** could wo with a single readLines or in blocks
con <- file("stdin", open = "r")
while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
  line <- trimWhiteSpace(line)
  words <- splitIntoWords(line)
  ## **** can be done as cat(paste(words, "\t1\n", sep=""), sep="")
  for (w in words)
    cat(w, "\t1\n", sep="")
}
close(con)
```

Fig 15. Mapper Code in R



```
#! /usr/bin/env Rscript

# reducer.R - Wordcount program in R
# script for Reducer (R-Hadoop integration)

trimWhiteSpace <- function(line) gsub("(^ +)|( +$)", "", line)

splitLine <- function(line) {
  val <- unlist(strsplit(line, "\t"))
  list(word = val[1], count = as.integer(val[2]))
}

env <- new.env(hash = TRUE)

con <- file("stdin", open = "r")
while (length(line <- readLines(con, n = 1, warn = FALSE)) > 0) {
  line <- trimWhiteSpace(line)
  split <- splitLine(line)
  word <- split$word
  count <- split$count
  if (exists(word, envir = env, inherits = FALSE)) {
    oldcount <- get(word, envir = env)
    assign(word, oldcount + count, envir = env)
  }
  else assign(word, count, envir = env)
}
close(con)

for (w in ls(env, all = TRUE))
  cat(w, "\t", get(w, envir = env), "\n", sep = "")
```

Fig 16. Reducer Code in R

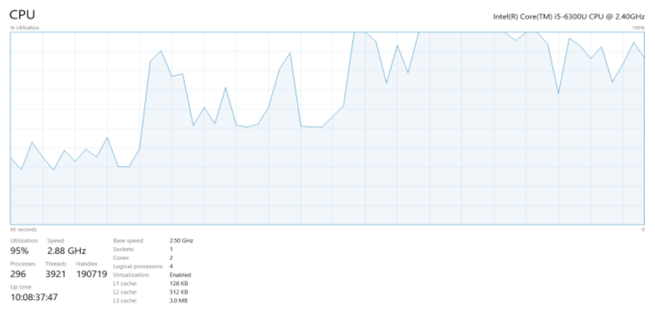


Fig 17. CPU Utilization for Hadoop MapReduce

The above figure is the CPU utilization of the machine when the Hadoop MapReduce was functioning. As it can be seen in the picture the utilization was approximately 100% throughout the process.

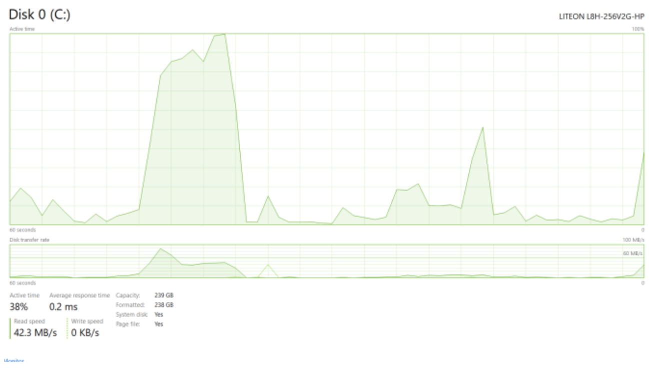


Fig 18. Disk I/O Utilization for Hadoop MapReduce

The above figure is the Disk I/O utilization of the machine when the Hadoop MapReduce was functioning. It can be noted that quite more memory was used as compared with Spark system.

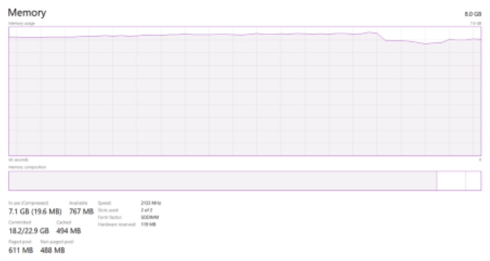


Fig 19. Memory Utilization for Hadoop MapReduce

The above figure is the Memory utilization of the machine when the Hadoop MapReduce was functioning. The memory was almost 100% throughout the process.

B. *SPARK with 'R'*

Apache Spark was an attempt made at UC Berkley to have something better than MapReduce. It is known for its in memory performance. 'Sparklyr' is a package from R to get the features of Spark with the ease of R. It is an interface between R and Spark and compatible with all the CRAN packages of R. For beginners of Spark, this library helps install Spark and set it up. This library has all the features of Spark. So after the communication is setup between R and Spark, one can simply work with the core nodes of Spark.

R faces a limitation of memory available for a single machine, but with 'SparkR' analyzing distributed data becomes easier. 'SparkR; provides the merits of both spark and R.

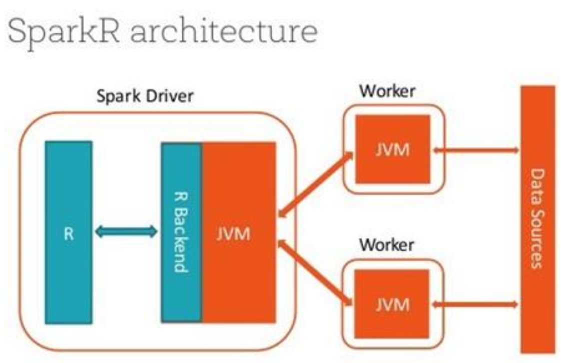
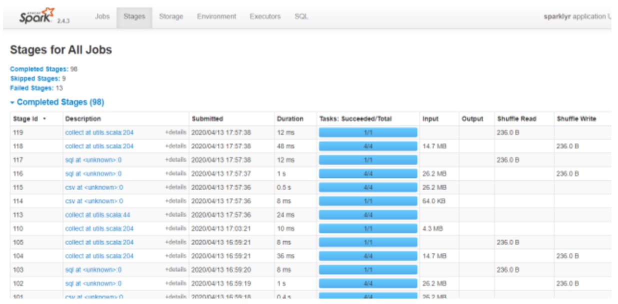


Fig 20. Spark Architecture

The above figure depicts the binding of R to JVM, which helps the R program to run jobs on all the available nodes on the cluster.



Stages for All Jobs

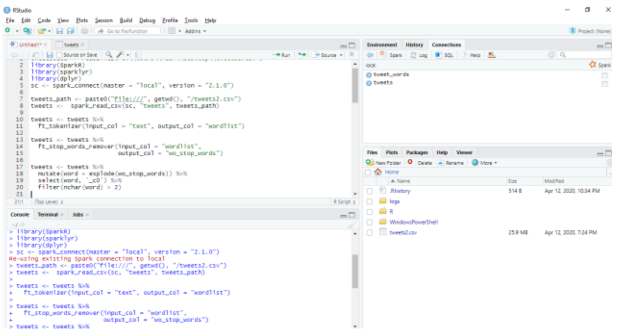
Completed Stages: 98
Skipped Stages: 9
Failed Stages: 13

Completed Stages (98)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
119	collect at utils.scala:204	+details	2020/04/13 17:57:38	12 ms	1/1			236.0 B	
118	collect at utils.scala:204	+details	2020/04/13 17:57:38	48 ms	4/4	14.7 MB			236.0 B
117	sql at <unknown>:0	+details	2020/04/13 17:57:38	12 ms	1/1			236.0 B	
116	sql at <unknown>:0	+details	2020/04/13 17:57:37	1 s	4/4	26.2 MB			236.0 B
115	csv at <unknown>:0	+details	2020/04/13 17:57:36	0.5 s	4/4	26.2 MB			
114	csv at <unknown>:0	+details	2020/04/13 17:57:36	8 ms	1/1	64.0 KB			
113	collect at utils.scala:44	+details	2020/04/13 17:57:36	24 ms	4/4				
110	collect at utils.scala:204	+details	2020/04/13 17:03:21	10 ms	1/1	4.3 MB			
105	collect at utils.scala:204	+details	2020/04/13 16:59:21	8 ms	1/1			236.0 B	
104	collect at utils.scala:204	+details	2020/04/13 16:59:21	36 ms	4/4	14.7 MB			236.0 B
103	sql at <unknown>:0	+details	2020/04/13 16:59:20	8 ms	1/1			236.0 B	
102	sql at <unknown>:0	+details	2020/04/13 16:59:19	1 s	4/4	26.2 MB			236.0 B
101	csv at <unknown>:0	+details	2020/04/13 16:59:18	0.4 s	4/4	26.2 MB			

Fig 21. Spark Workflow Stages

The above figure shows the functioning of the word count and generating wordcloud using ‘R’ and ‘RStudio’



```
library(SparkR)
library(sparklyr)
library(dplyr)
sc <- spark_connect(master = "local", version = "2.1.0")

tweets_path <- paste0("file:///", getwd(), "/tweets2.csv")
tweets <-  spark_read_csv(sc, "tweets", tweets_path)

tweets <- tweets %>%
  ft_tokenizer(input_col = "text", output_col = "wordlist")

tweets <- tweets %>%
  ft_stop_words_remover(input_col = "wordlist",
                        output_col = "wo_stop_words")

tweets <- tweets %>%
  mutate(word = explode(wo_stop_words)) %>%
  select(word, `_c0`) %>%
  filter(nchar(word) > 2)
```

```
> library(SparkR)
> library(sparklyr)
> library(dplyr)
> sc <- spark_connect(master = "local", version = "2.1.0")
Re-using existing Spark connection to local
> tweets_path <- paste0("file:///", getwd(), "/tweets2.csv")
> tweets <-  spark_read_csv(sc, "tweets", tweets_path)
>
> tweets <- tweets %>%
+   ft_tokenizer(input_col = "text", output_col = "wordlist")
>
> tweets <- tweets %>%
+   ft_stop_words_remover(input_col = "wordlist",
+                         output_col = "wo_stop_words")
> tweets <- tweets %>%
```

Fig 22. Spark Code

Here is the code implementation of wordcloud for Spark in R.

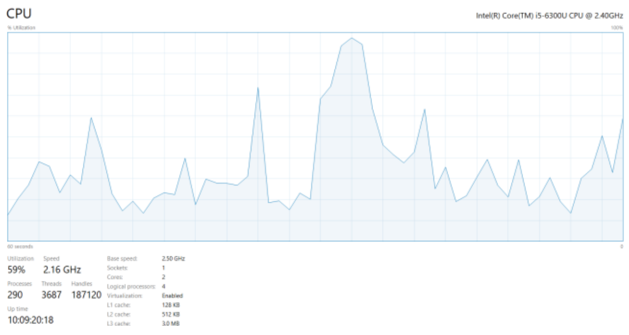


Fig 23. CPU Utilization for Spark

The above figure is the CPU utilization of the machine when the Spark was functioning. As it can be seen in the picture the utilization far lower as compared to Hadoop MapReduce.

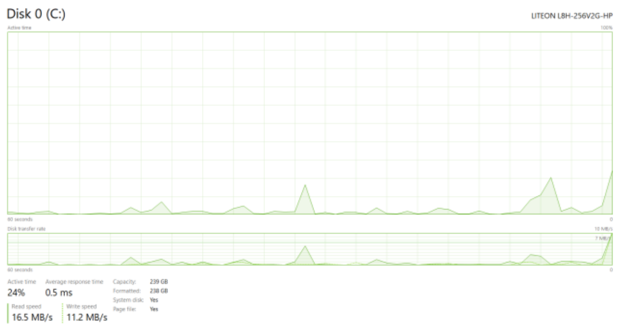


Fig 24. Disk I/O Utilization for Spark

The above figure is the Disk I/O utilization of the machine when the Spark was functioning. There was hardly any load to machine.

The Memory utilization of the machine when the Spark was functioning very low comparing with Hadoop MapReduce.

V. EVALUATION

As the final objective was to obtain the resource comparison between Hadoop MapReduce and Spark by performing data analysis forming a word cloud. Below is the word cloud obtained by both the systems.

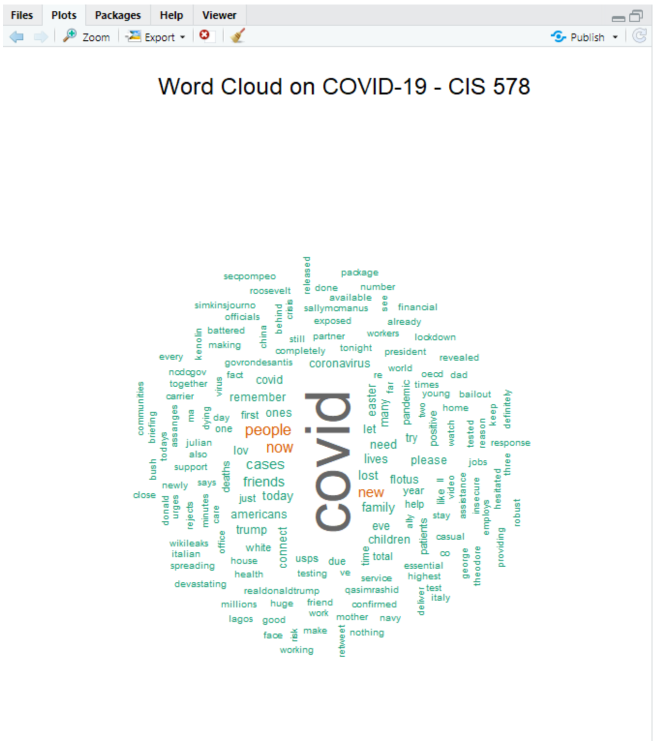


Fig 25. WordCloud obtained by Hadoop MapReduce and Spark

Spark is 10 times faster than Hadoop as Spark stores data in RAM and Hadoop on disk Hadoop does only batch – processing whereas spark can do both batch processing and real time processing.

Hadoop being disk-based it asks for disk those are fast for processing large amount of data while Spark only needs huge RAM and can work with standard disks. Also, Spark is easier to use as it needs less amount of code. For every ten lines in Hadoop MapReduce there are only two lines for Spark programming.

VI. Discussion

We have taken the paper – 'Performance Analysis of NoSQL Databases Having Hadoop

Integration' as reference that has a detailed discussion on various types of NoSQL databases and its integration with Hadoop. The authors have also performed evaluation on which combination is more suitable for real time applications by taking the parameters such as performance, scalability and fault- tolerance into consideration[1].

Also, the discussion and the design goals from the paper – 'SparkR: Scaling R Programs with Spark' is what intrigued us towards this project. The paper is about an R package that enables large scale analysis of data using the computation engine of Spark. It also gave us an insight on how the Data Frame API is used for computation that is scalable[8].

VII. Challenges

We had difficulty getting API and Access token details from twitter. Therefore we created the Twitter Developer account and have to request API credentials by creating an application.

Downloading 1M twitter data was another arduous task because twitter stores only 7 days of data and also we faced challenges while loading data to CSV file as 1M data cannot be loaded at once and we had to download it in smaller chunks.

We also posed several challenges while installing packages in R on Windows machines.

Performance and installing issues when we were integrating RStudio with Spark and Hadoop.

VIII. Conclusion

We are successful in obtaining the unstructured raw data from social media sites and store them on the Hadoop and Spark systems, processed the data using MapReduce using RStudio.

Furthermore, we have analyzed the resource utilization on both the architecture system, such as CPU utilization in terms of memory and time. This implementation helped us in understanding the MapReduce operation on Spark as well as Hadoop, also making us familiar with its infrastructures.

IX. Acknowledgment

We would like to thank Prof. Roy for his involvement and lectures. His continued support made this implementation easier for us.

X. References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T.

Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

[8] Venkataraman, S. (n.d.). SparkR: Scaling R Programs with Spark. Retrieved from https://cs.stanford.edu/~matei/papers/2016/sigmod_sparkr.pdf