MAP REDUCE DATA PARTITIONING

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

By

Aashrita Goel, 141290

Under the supervision of

Dr. Suman Saha

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234,

Himachal Pradesh

Candidate's Declaration

I hereby declare that the work presented in this report entitled **" Map Reduce Data Partitioning"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from August 2017 to April 2018 under the supervision of **Dr. Suman Saha,** Assisstant Professor, Dept of CSE, JUIT**.**

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Aashrita Goel, 141290

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Suman Saha
Assisstant Professor(Senior Grade)
Department of CSE and IT

Dated:

ACKNOWLEDGEMENT

It is great pleasure to express my gratitude and words of appreciation to the people who have been, in various ways, the source of help, inspiration and encouragement in my life. I express my heartfelt gratitude to my guide **Dr. Suman Saha** for giving me an opportunity to work on this project. I am also thankful for the constant guidance and motivation that he has put in. It has been a good learning experience to work with him. I am thankful to all the members of JUIT for their support in completing this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Data clustering is the grouping of similar data points in a cluster. Data clustering helps a lot in data analysis of big data. Map reduce framework is used for the parallel processing of big data. Map Reduce functionality can be used to implement clustering algorithms to cluster such huge amount of data. In this project, one modified Map Reduce clustering algorithm has been proposed that is computationally less expensive than the k-means via Map Reduce algorithm. The performance of the traditional k-means, k-means via Map Reduce and the modified clustering algorithm was compared. The proposed algorithm came out to be better than the k-means via Map Reduce, given that the dataset for clustering is huge. For small dataset, serial k-means performs better than the Map Reduce clustering algorithms. For large dataset, k-means takes a lot of time.

CHAPTER- 1

INTRODUCTION

1.1. Introduction

With the advancement of technology, tons of data is generated that can no longer be managed and processed with the help of traditional file systems. This led to the introduction of the term “Big Data”. Big data refers to large datasets that are so voluminous and complex that new technology, storage system, and processing unit had to be developed to handle it[1]. For the analysis of such big data, data clustering helps a lot.

Data clustering is the partitioning of a data set or sets of data into similar groups. It is a common technique used for the purpose of data analysis and has many applications in the field of statistics, data mining, and image analysis. In data clustering, we group the objects similar to one another and different from other groups. For this purpose, some attribute is used to identify the objects similar to each other that can be placed together in a group. In our study, distance measure is used for data clustering. There are many types of data clustering algorithms which include hierarchical algorithms, Lloyd’s algorithm etc. Hierarchical algorithms build successive clusters using previously defined clusters. Hierarchical algorithms can be agglomerative or decisive. Agglomerative clustering algorithm follows a bottom up approach in which clusters are built by successively merging smaller ones. In divisive clustering algorithms, clusters are formed by splitting larger cluster and thus this approach is known as top-down approach of clustering.[2]

Data clustering is computationally expensive in terms of both time and space complexity. In addition more time and space is consumed when precision and accuracy has to be met in terms of similarity within the data clusters. The situation becomes worse when the data is distributed. Hence, parallelizing and distributing expensive data clustering tasks becomes cumbersome. A good knowledge of parallel and distributed programming concepts are needed to carry out the data clustering tasks in such cases with great efficiency.

Apache Hadoop is an open source, java-based programming framework that helps in the storage and processing of big data.[2] MapReduce is the software framework for solving certain kinds of distributable problems that involve big data. It is a two step process that consists of Map and Reduce phase. Both the mapper and reducer phase take the input in the form of (key, value) pair. The input file is given as input to the mapper phase which produces the intermediary output which is given as input to the reducer phase. The reducer phase gives the final output which is written in the file. Mapper phase includes multiple mapper tasks and each map task processes its part of the problem and outputs result as key-value pairs. The reduce step receives the outputs of the maps, where a particular reducer will receive only map outputs with a particular key and will process those.

One of the biggest advantages of MapReduce is that Map and Reduce tasks can be distributed across different nodes. Hence, MapReduce provides a platform for distributed computing framework. Mapper and reducer both can run independently. Hadoop provides a number of features that make it suitable to process big data. Among these features is reliability that is achieved by replication and also provides the data lost in case of node failures and other disasters.

The common data clustering algorithm implemented in this study is K-means, and moving forward we will be implementing a modified MapReduce clustering algorithm. Calculations required for k-means are computationally costly and perform well for data having the accompanying three qualities:

- relatively low feature dimensionality,
- limited number of clusters, and
- a modest number of data points

In this study we explore k-means application through MapReduce using Hadoop. The basic approach is to form clusters initially by random initialization and then by deliberate initialization and moving forward Spectral analysis can also be applied. This could be one of the future developments.

Chapter two explains the K-means clustering algorithm that we implement using MapReduce. Chapter three discusses Hadoop which is the MapReduce implementation we used. In this chapter we also detail the MapReduce strategy we use to tackle this particular data clustering problem.

Chapter five summarizes the report and provides details on future directions.

1.2. Problem Statement

The time complexity of existing data clustering algorithms to cluster large datasets is huge and in Lloyd's algorithm, distance is calculated in every iteration to find the new centroids which makes it computationally very expensive. Thus, to optimize the traditional algorithms Moreover, the clustering algorithm has to be optimized.

1.3. Objective

The objective of the project is to utilize the Map Reduce structure on an extensive dataset to distinguish the groups inside that dataset by forming *k* clusters.. The objective also includes comparative study between the traditional k-means, k-means++, k-means via Map Reduce and the modified Map Reduce clustering algorithm.

1.4. Methodology

For this project, we followed the following steps:

Step 1. Data Collection: It is the first and primary thing that is to be done so that we can apply clustering algorithms over it. Data is randomly generated and large data set is taken from the STARNET lightning network.

Step2. Load data on HDFS: The data set collected is put on the Hadoop distributed file system.

Step3. Implement clustering algorithms: Serial k-means, k-means++ and modified map reduce clustering algorithm is designed in python and applied on the data collected.

Step4. Store the result: The final clusters and time taken to implement the algorithms are written down to the output directory.

Step5. Plot the graphs: The clusters so formed are represented using clusters and the final centroids are also plotted.

1.4.1. Data Collection:

Several data samples have been generated randomly or collected to perform MapReduce algorithm over it. Data samples used for the purpose are as follows:

- **Randomly generated data set:** This dataset is generated randomly using numpy module of Python. For each instance, there exists three attributes, one for the data point identifier and other two attributes determine the two dimensions.

Data Set Characteristics:	N/A	**Number of Instances:**	1126
Attribute Characteristics:	Integer	**Number of Attributes:**	3
Associated Tasks:	Clustering	**Missing Values?**	N/A

Table 1.1 Randomly generated data set

- **Thunders data set:** This data set represents the STARNET detection network in which clustering is based on the distances of longitude and latitude attribute of data points.

Data Set Characteristics:	N/A	**Number of Instances:**	10788
Attribute Characteristics:	Integer, Float	**Number of Attributes:**	29

Associated Tasks:	Clustering	**Missing Values?**	N/A

Table 1.2 Thunder Data set [3]

1.4.2. MapReduce Algorithm:

Map Reduce is the programming model of Hadoop that is used for the analysis and processing of big data. To perform spatial clustering(clustering based on the distance metric) over the data collected, we implemented mapper and reducer algorithms of k-means and k-means++ in Python. An advancement in the mapper and reducer phase was made and implemented. The input data goes through the following stages before writing the final output in the output directory:[4]

1. **Input:** The input directory that contains the large datasets is given as input to the MapReduce task. In this stage, input data is split into many independent data blocks that are given as input to multiple mappers for parallel processing.

2. **Map:** It takes the input in the form of <key, value> pairs. Thus, the input phase is also responsible for converting the input data in the <key, value> pair form. Map functionality is then applied to each of the <key, value> pair. Map phase produces the intermediate result in the <key, value> pair form which is fed in as input to the next stage.

3. **Shuffle:** In this phase, the <key, value> pairs that are given as output by the Map stage are sorted on the basis of the key and the <key, value> pairs that have the same key are merged in the form <key, list of values>.

4. **Reduce:** This phase will iterate through the data points present in the list of values assigned to a particular key. The input is given in the form < key, {a list of values} > and the output is in the form < key, value > pair.

5. **Output:** This stage will write the results of the Reducer stage to the specified output directory.

Operation mechanism of MapReduce is shown in Figure 1.1.

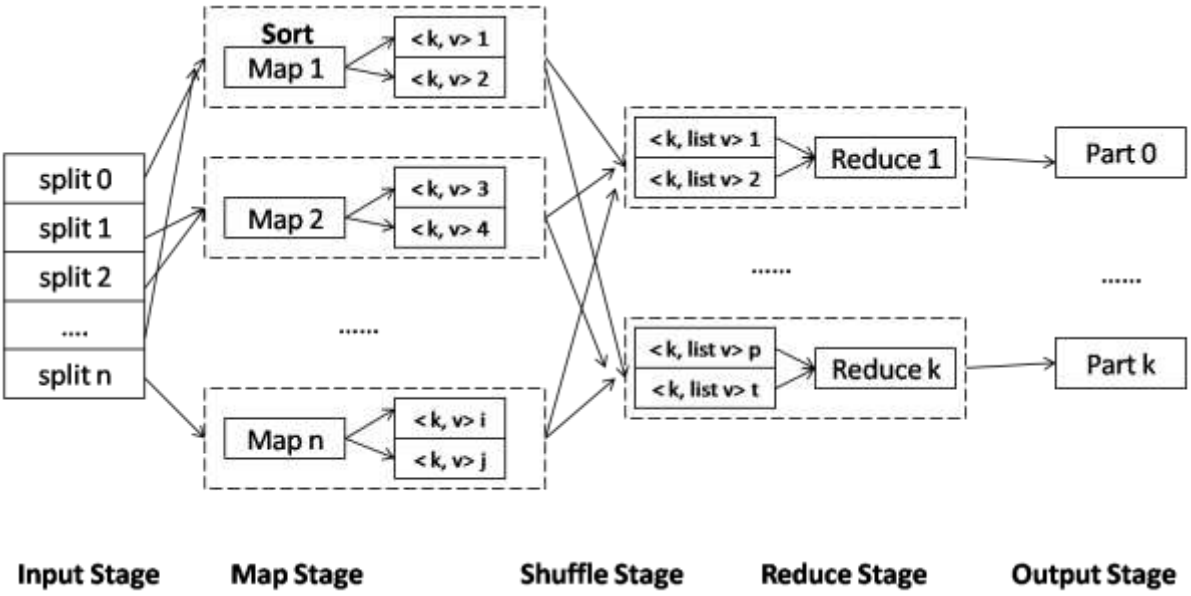


Figure 1.1: Operation of MapReduce

CHAPTER 2

LITERATURE SURVEY

2.1. Hadoop and Big Data

S. Vikram Phaneendra & E. Madhusudhan Reddy enlightened us by explaining the change in data and the way we can manage it. Initially, the amount of data generated and to be handled was less and thus could be easily managed and analyzed using RDBMS. Also, we had only structured data mainly that could be easily converted to relational databases. But, these days the data available might have a proper defined schema i.e. the Structured data or might be semi-structured or unstructured like audio clips, images etc. Also, the amount of this generated data is too high and is thus termed as Big Data.[1] The 5 dimensions of Big Data that differentiates it are :

Figure 2.1: 5 V's of Big Data

i. **Volume**: Big Data as the name describes refers to the huge amount of data.
ii. **Velocity**: It refers to the speed at which this data is being generated and the speed required for storing and analyzing this huge amount of data.
iii. **Variety**: It refers to the diverse content of data as we discussed earlier, it might be a mixture of structured, semi-structured and unstructured data.

iv. **Veracity**: It refers to the use of the generated data. The reliability of the data that we are storing is also an essential factor.

v. **Value**: It refers to the useful information that we can obtain from this data. When we are ready to invest so much in storing and analyzing this huge amount of data then, we must get an output that would be beneficial for us.[1]

Bernice Purcell stated that the huge datasets that are present in Big Data cannot be handled using simple relational databases and traditional RDBMS tools. Instead, we need a clustered Network Attached Storage for storing this huge amount of data. Hence, we needed something for proper storage and analysis of this data and the solution is Hadoop. The Hadoop architecture can be used for storing this huge amount of data and using various techniques we can extract meaningful information out of this structured, semi-structured and unstructured data. [5]

Hadoop can be referred as a collection of many open source software utilities. It is basically a software platform that can be used to store high volume of data and perform large number of computations on that data in a very easy and precise way. Today almost every company uses it for both Research as well as for Production. Hadoop can be analyzed as an ecosystem consisting of all open source elements that brings out the fundamental changes in the way organizations work. It actually facilitates us by providing a large network of computers where we can store huge amount of data and perform all the programming operations depending upon the requirement we need. While designing the various modules of Hadoop the major consideration was the high failure rate of commodity hardware and thus it should be automatically handled by the software in the framework. It uses the Map Reduce programming model for performing all the computations and processing.

This framework was originally designed for computer clusters that were made out of commodity hardware and thus, all the modules of the framework were designed in accordance to the high failure rate of commodity hardware. If we look at the core of Apache Hadoop, the storage part it consists of is known as HDFS i.e. Hadoop Distributed File System and the Map Reduce Programming model as a part of processing. Now considering the fact that Hadoop is a platform where we can deal easily with Big Data, we have to know

how it actually works. Hadoop splits the data files into large blocks and distributes them across the data nodes(systems) in a cluster. To process these data blocks on different machines, the packaged code is transferred to the data nodes(systems). In this way, huge datasets can be processed in a faster and more efficient way.

Java Programming Language has been used for developing Hadoop. Although, java is the most common language for Hadoop Map Reduce, but we can use any programming language along with Hadoop Streaming to implement the Map Reduce programming model.

Various modules of Apache Hadoop Framework are:

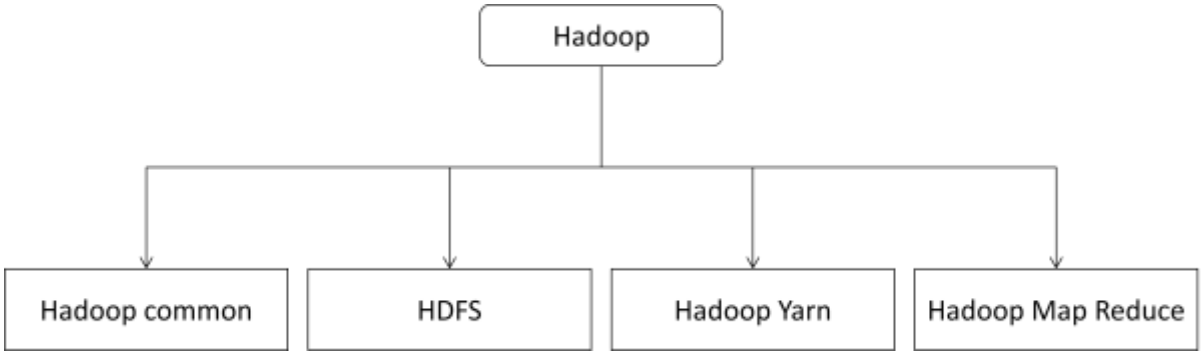


Figure 2.2. Modules of Apache Hadoop

Hadoop Common: It consists of all the libraries and utilities that are required by other modules and thus acts as Credential Provider. It is basically a Key Management Software.

HDFS: The Hadoop Distributed File System lays its basis from Google File System, it is a distributed file system that stores all the data on commodity machines, thereby providing very high bandwidth across the clusters.

Hadoop YARN: It manages the computer resources in clusters.

Hadoop Map Reduce: The programming model for large scale data processing in Hadoop.
Now let us explore Hadoop Architecture in detail.

Hadoop Architecture mainly explains how the file system in hadoop works and how the Map Reduce is used to process the large amount of data stored in these file systems.[1]

2.2. HDFS

The Hadoop Distributed File System is a portable as well as scalable file system written in java programming language. Harshawardhan S. Bhosale and Prof. Devendra P. Gadekar explained that Hadoop Distributed File System basically operates on the clusters of computers. Hadoop itself forms the clusters of computers and it is these clusters that HDFS lies its basis on. If we consider a cluster, it consists of many nodes. There are two types of nodes within the cluster, the name node and the data node. There is only one name node i.e. the node that contains all the meta data and is concerned with the namespace of the file system. As the name suggests data node consists of large blocks of data.[6]

Now when data storage is a problem Hadoop Distributed File System is there to hold your large amounts of data varying from few gigabytes to terabytes with very high efficiency. It stores large files across multiple machines that are termed as Data nodes. It is these data nodes that serves all read and write requests of the clients. Initially the large files are broken into various blocks depending on their sizes. Each block is then stored at multiple places(Data Redundancy) so as to achieve high reliability. It is because of this replication of data across multiple hosts that (RAID) Redundant Array of Independent Disks is not required. The default replication value is 3 i.e. three copies of one data block are stored out of which two are stored on different machines within the same rack while third on some other rack. Also data nodes can communicate with each other to maintain the balance of data and thus keeping high reliability.

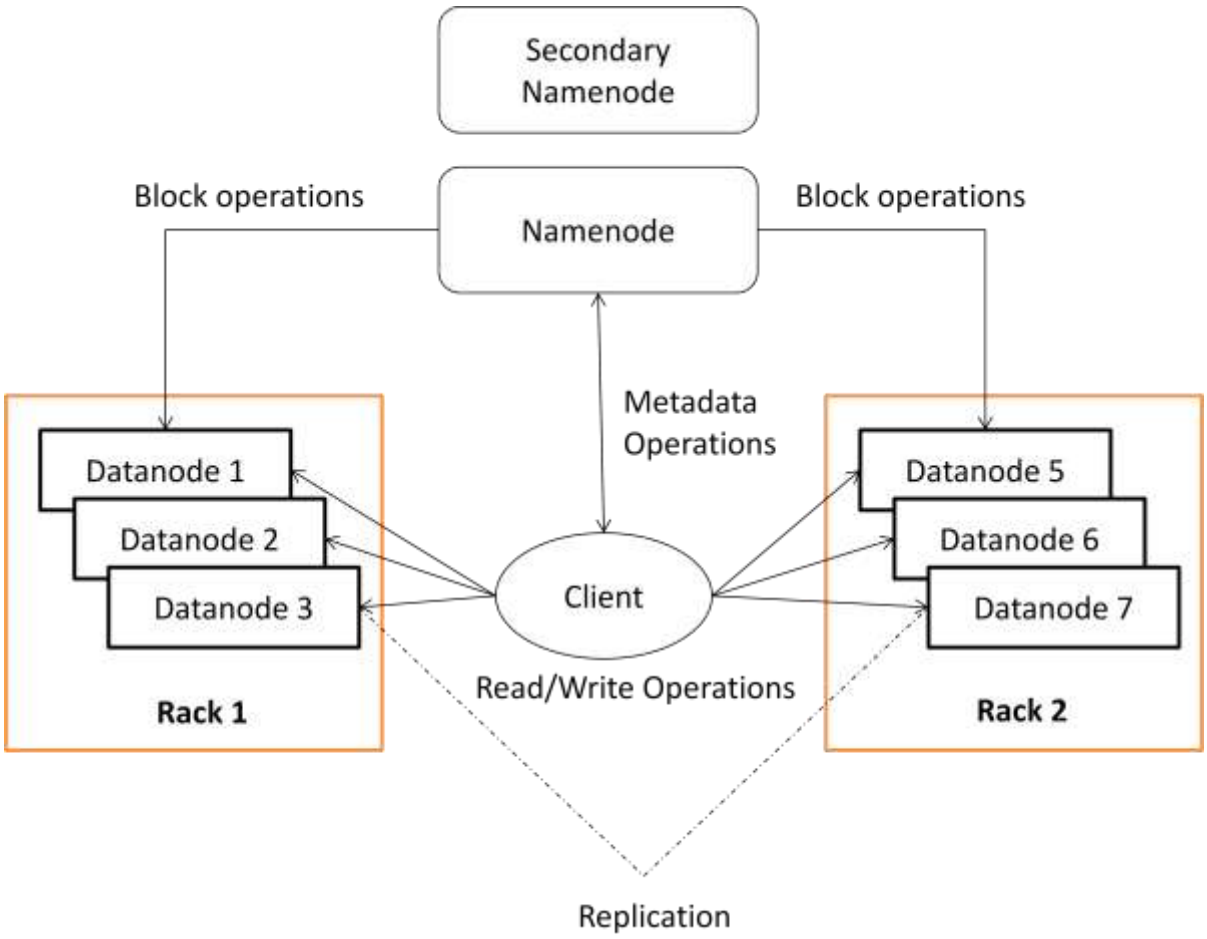


Figure2.3: Architecture of HDFS[7]

But HDFS strictly follows - "Write once Read Many" model. Hence, we cannot write again and again instead, we can just open a file and append more data to it. But for this append operation as well as read operation the client needs to request for the access from the name node. [7] Once the name node provides the client all privileges then only one can perform read and write operations block by block. As we know, HDFS follows master slave paradigm- name node being the master and data nodes the slaves.

To perform a HDFS write operation, client needs to request for the access from the name node then it is the name node that provides the address of the data node to the client where client can perform write operations. Once the write operations start, data pipeline is created

by the data node and the blocks are copied to other nodes also for maintaining the replica rate.

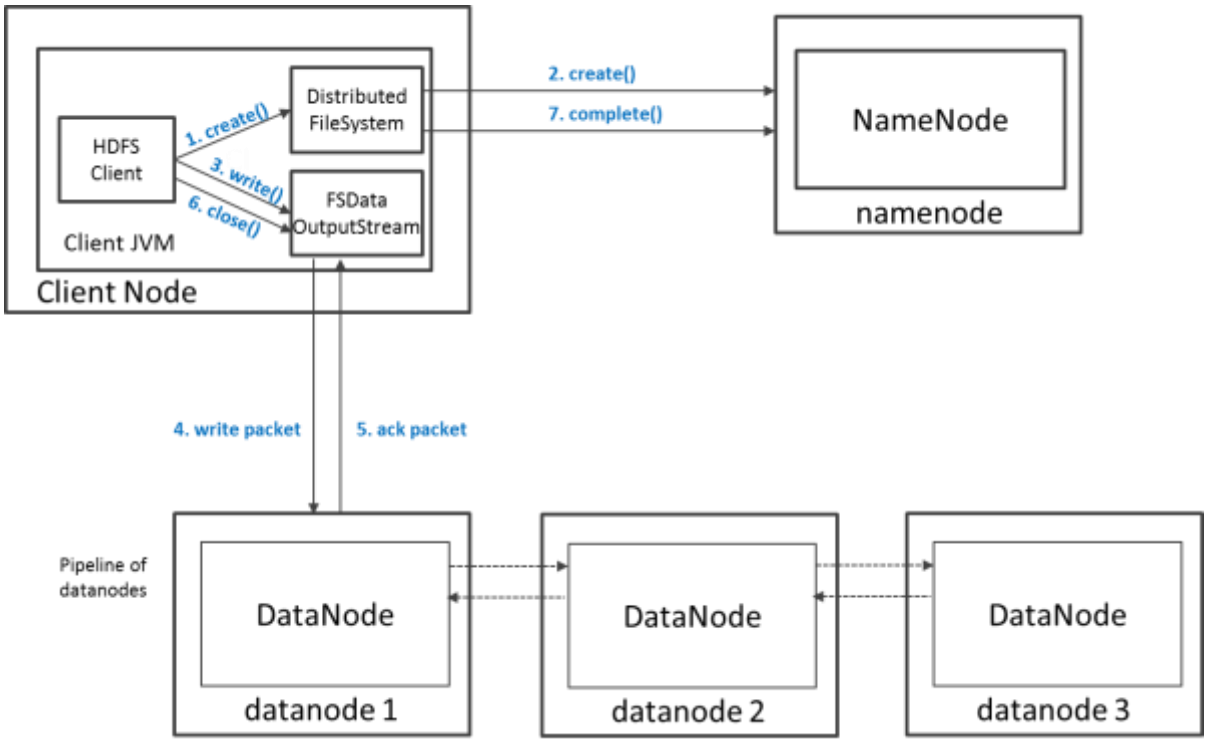


Figure2.4: HDFS write operation[8]

Each cluster consists of a particular primary name node and the secondary name node that are interconnected. The primary name node contains all the meta data i.e. the data about the data stored in data nodes and location of all the blocks etc. But if the situation arises that the primary name node fails, this is where secondary name node comes into action. The secondary name node on regular intervals take snapshots from the primary name node and stores them to local directory so that if primary node fails at any point of time these images could be easily used to restart a primary node again. Since the name node is the single point of storage of meta data so, the risk of this huge loss cannot be taken in today's world, this could turn as a huge bottle neck in the storage management of an organization. Although snapshots are taken periodically and saved but still true redundancy is not provided by the secondary name node.

To perform a HDFS Read operation client needs to request for the access from name node only then name node checks for the particular privileges, and then provides the address of the data node to the client from where client can directly read the block. Then, the client is able to access the data nodes directly where the blocks of file are already stored. [9]

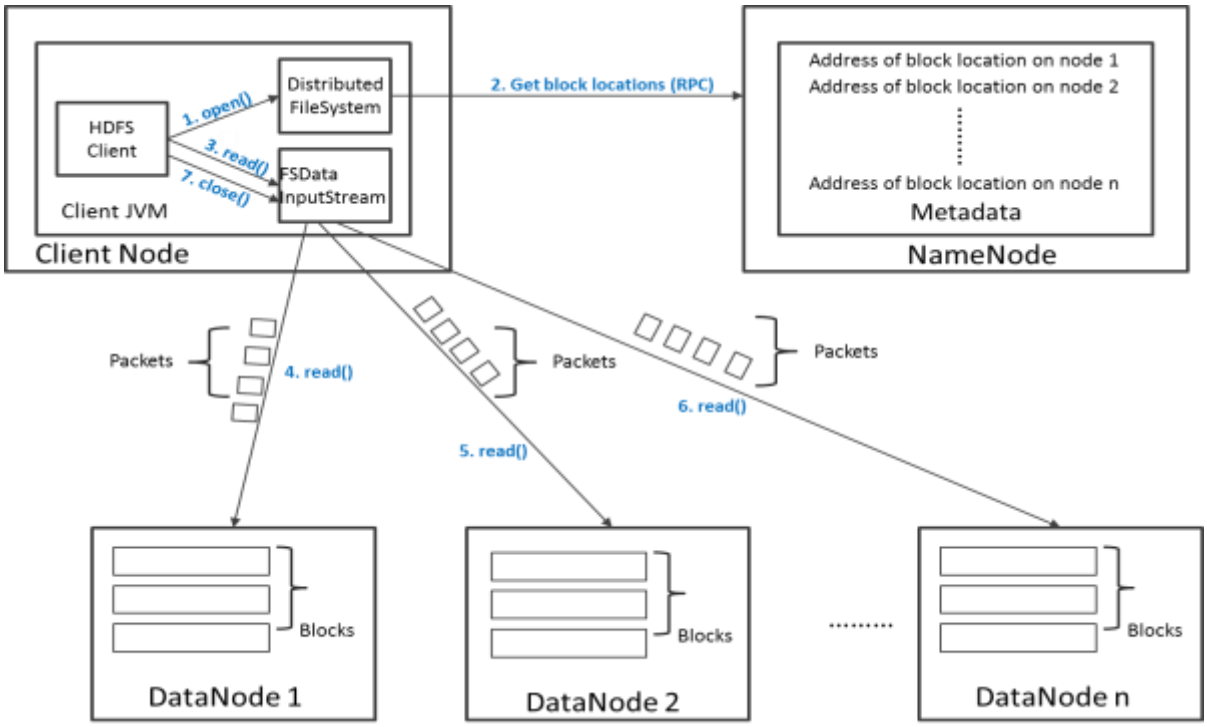


Figure2.5: HDFS read operation[8]

The main basis in Hadoop is to get higher throughput. Hence, it considers the high cost of collecting data at a single place performing all the computations there only and then transferring the results to the various other nodes in the whole network. So, instead of performing such high cost tasks the idea is to move the application closer to the data for performing all the required computations i.e. either on the same data node or on a node within the same rack thereby reducing the costs and increasing the throughput.

Hadoop also facilitates us by providing Hadoop Streaming. It is Hadoop streaming that makes it possible to run any executable as a mapper and a reducer for performing all the computations.

2.3. Map Reduce Engine

Now, as we know HDFS is used for storing the large amount of data, but we also need to perform computations on this data and thus, we have the Map Reduce paradigm. The Map Reduce engine contains a job tracker. Whenever a client submit a map reduce job to the application, it reaches out directly to the job tracker. Also, we have the task tracker nodes to which the tasks are assigned by the job tracker. Now the main priority of job tracker is to assign the tasks to be performed and it knows the location of machines where the data to be computed resides and the machines nearest to that machine within the same racks and the same clusters, so the job of the job tracker is to assign the tasks in such a way that computations should be performed closer to the data as much as possible. If it is not possible to assign the work to the same node on which data resides then it is assigned to the machine within the same rack. The reason behind the whole task is to keep network traffic as low as possible. But, there is always a single job tracker where as Task trackers are multiple. Also, each node has the potential to act as a slave task tracker.

If the task tracker is not able to fulfill the job in time or if it fails, that particular part of job can be rescheduled. Also, the job tracker can keep a record of number of available slots for a particular task tracker nodes in the cluster. Whenever a map or reduce task is activated, one of these slots is filled up. Thus the job tracker allocates the job to the nearest task tracker where at least one slot is available. But it does not consider the actual load of the system and hence the real availability is not known. Sometimes even the task tracker might be too slow and thus might delay the map or reduce job or even both. But for such cases we can even execute this particular task on multiple slave nodes.

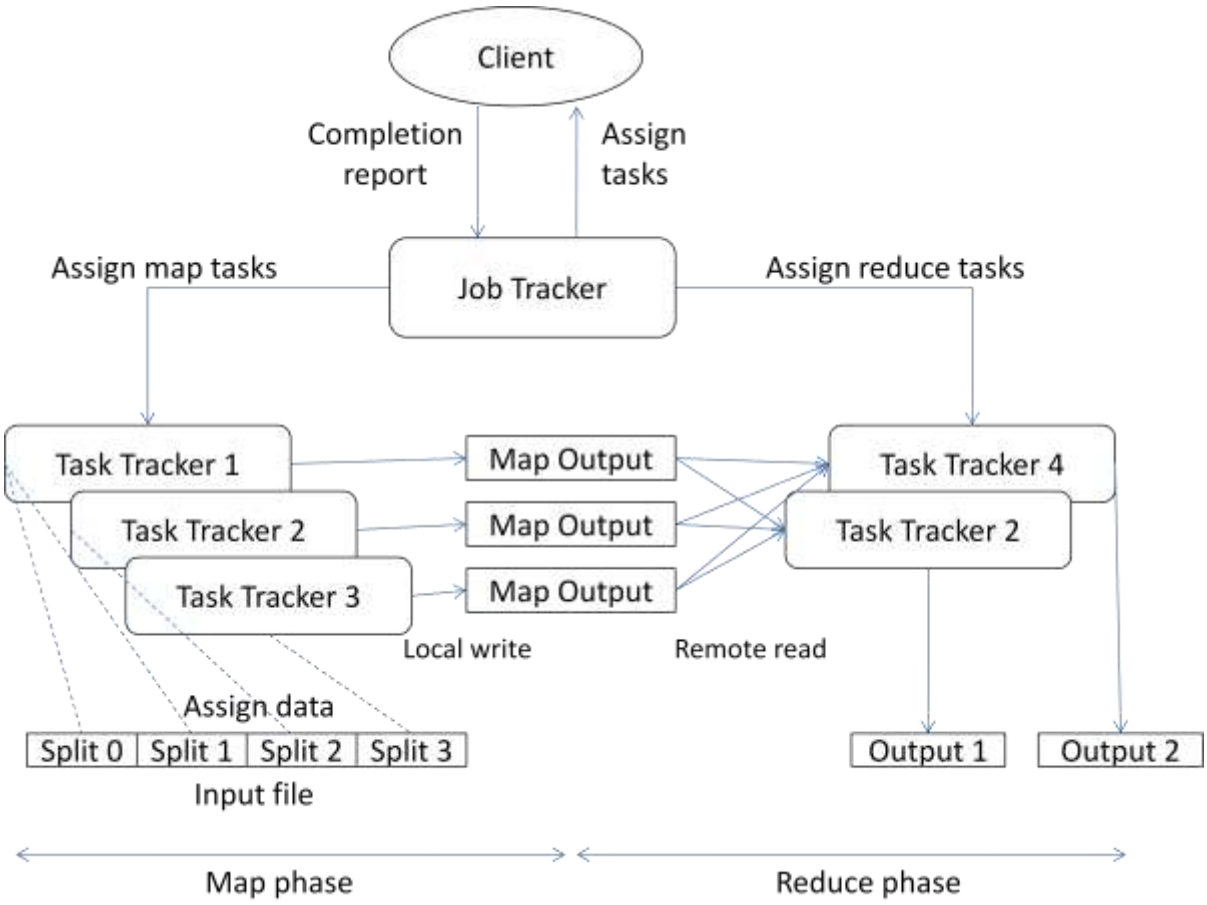


Figure 2.6: Architecture of Map Reduce

Jeffrey Dean and Sanjay Ghemawat explained that Map Reduce is an implementation for the process of generating and managing large data sets. It normally consists of two functions i.e. mapper and the reducer. It is these functions that perform the various computations and give us the desired results. It is the user only that defines both mapper as well as reducer for the computations to be performed. Map step takes input data and generates intermediate key value pairs and these key value pairs are taken as input via reduce function that performs computation on these pairs and gives us the desired results. We can consider map and reduce to be as separate steps on data that can occur in parallel. Although the result of mapper is used by reducer still they are executed multiple times in parallel. [10]

For instance, a huge amount of data can be easily and efficiently reduced to smaller chunks of data where we can apply data analytics easily. When we are working with Hadoop such

operations are performed as map reduce jobs. Once these operations are performed, we can write the desired results back to HDFS. So basically, we have two functions in MapReduce:

- **Map function**: It takes a section of data values as an input, perform the desired operation on each input thereby generating an intermediate output. The output is in the form of key value (key, value) pairs.

  Once the map step is over, the outputs from the map step are sorted and it is taken as input in reducer. In betwen, the combiner might also be used. The data transfer takes place from mapper to reducer. All the values belonging to a particular key from the output of mapper function are aggregated at a single node where reducer for that particular key has to be executed.

- **Reducer function**: The reducer function then takes this aggregated values for a particular key as an input and finally generate a key value pair as a desired output for all particular keys i.e. if we consider the intermediate key value pairs generated by the mapper function many key value pairs were generated with non unique keys but then, the similar keys were aggregated at a single node where reducer performed the necessary computations to generate a single output for a single key which demarcates our desired result.[11]
  Thus, when it comes out to be dealing with Big Data and its analysis, Map reduce is the best suited.

2.4. Hadoop Streaming

Hadoop distribution has hadoop streaming as one of its utilities. We can create and run map as well as reduce jobs with any executable as a mapper or reducer with the help of this utility. The utility creates the job and assign it to a specific cluster where it could be completed thereby generating the appropriate result. [12]
In hadoop streaming, the number of mappers assigned to do the task depends on the number of data points. The number of reducers is also chosen by the hadoop streaming by default. However, user can specify the number of reducers.

By using hadoop streaming, we can write mapper and reducer code in any programming language.

2.5. k-means Clustering Algorithm

As we discussed above, to handle Big Data we need Hadoop and thus we need to organize the machines in the forms of clusters and each system serves as data point in the cluster. Before all this we need to partition the data sets also and this partitioning of data sets into clusters is required in most of the applications these days. We have many clustering algorithms for partitioning of data sets. The most common and widely used algorithm is Lloyd's algorithm. Lloyd's algorithm is the simplest algorithm for clustering of data sets as it just requires one single input k i.e. the number of clusters required and thus commonly known as K-means algorithm. The complexity is another factor that makes this algorithm so useful.[15]

The k-means algorithm and its other variants can also be implemented using map reduce and thus get its importance. Let us suppose we have n objects in our data set, then k-means is basically implemented to partition these n data points into k clusters and this partitioning is done based on both similarities and differences within the data points. The clusters are made based on similarities i.e. within the cluster all the data points have a common trait and thus have a high intra cluster similarity. Instead, the inter cluster similarity is very low. Also, k-means sets a center of gravity for every cluster and this center of gravity depends on the number of objects inside the cluster. So basically, if there is any change in the objects present in the cluster, the center of gravity also makes a shift.

To begin with the algorithm, we have n data points and the input k i.e. number of clusters. Now we have to select k centers of gravity for k clusters, we select them out of n data points that we have. Now we consider all the other n-k data points and assign every data point to one of these k selected centers based on a similarity (to be chosen by user). When all the data points are alloted a center, we have with us k clusters but these are not the true centers of

gravity and even the clusters are not proper even. So, we need to perform various iterations to get the desired clusters and the perfect centers of gravity for each one of them.

Now we have all the clusters for once and thus we can move forward for the calculation of new centers of gravity. The above two steps have to be performed iteratively until and unless we get the desired result i.e. we reach out the convergence condition. Now, in k-means clustering comparing the similarity level is the most challenging task. Once the new centers gravity have been calculated, then we again check for all the data points for similarity with the data centers and if it matches with some other center of gravity then it is allotted that particular cluster.[16][17]

For calculation of assigning the objects their new desired clusters the algorithm has to perform around n*k calculations i.e. the distance computations. Although few of the steps like calculation of distances between data points and the new centers of gravity can be computed in parallel but we cannot go for all the iterations in parallel as new centers of gravity have to be calculated for every iteration. Since the centers of gravity i.e. the mean of all the clusters changes with each iteration so we cannot go for parallel computation haphazardly, thus it gets the name - "Serial K-means algorithm" .[13]

There are many implemented forms for K-means clustering but the most common is heuristic. It uses a refinement technique iteratively. Although it is just heuristic but sometimes it is mistaken as k-means clustering algorithm only.[16] In Lloyd's algorithm the data is divided into k sets using or method or even arbitrarily. Once we have formed the clusters for the first time i.e. for the first iteration by calculating the distance with these randomly selected centroids then we have to calculate fresh centroids for all the clusters. The fresh centroids can be calculated by computing the means of the clusters that have been formed in the previous iteration. After calculating these new means, we can associate all the data points with their nearest means by calculating distances. This is how there is a change associated with every iteration as new data points are added into some clusters while some data points are removed from some clusters. This algorithm is repeated again by calculating new centroids and the iterations continue. But as we know there must be an endpoint

associated with every algorithm, so we have to perform the iterations but only till we reach the convergence condition.[18][19]

When we go on iterating, there come a stage when centroids no longer change and therefore all the clusters will remain uniform after that, this is the phase of algorithm that we call as convergence condition. The convergence condition in Lloyd's heuristic is reached very quickly thus making it more popular.

But the input value k in the k-means algorithm is both friend and a foe. It is advantageous as we don't have to give so many inputs, just the desired number of clusters and thats enough for it. But at the same time, it turns out to be a foe for our systems as if we won't choose the appropriate value for k, it might give us poor results.

Moreover, the convergence condition can be set according to the requirement of the users. Basically, the convergence is when there is no total error in between the iterations. But, in some implementations the convergence condition can be modified to the situation when the total error drops below a certain threshold level. In k-means the time complexity depends on three factors: first is the number of data points is the data sets that have to be partitioned , second is the number of clusters to be made and third is the dimension of the data points.[20]

We can apply k-means using a distributed setting also. We can apply the distributed version using map reduce paradigm. As we know that map reduce paradigm works in Hadoop where we can split the total computations into two parts: using method map and using method reduce. So basically, we will be breaking each iteration into 2 steps first step would be applied using mapper function and the other one using the reducer function.

So basically the first step would compute the data points that are closest to the centroid. Thus, at the end of application of mapper function we would get the data points associated with each centroid. In the second phase i.e. the reducer phase we can simply compute the new centroid using the existing clusters. This completes our 1 complete iteration. This tells us the fact that we need to execute mapper and reducer function again and again until we reach the convergence condition.

For instance, if we have a data point x in the data set , the map phase applies on each data point x and generates the key value pair as output where key is the index of the mean i.e. nearest to x and value (x,1) is returned. The reduce phase operates on these key value pairs. For each key, one reducer works and does the pair-wise summation over all the values belonging to that particular key. But as we know, these iterations i.e. every mapper and every reducer has to be executed on a different machine, thus every machine must have the list of means that would be broadcasted to all of them after every iteration.

Here in the case of map reduce application we can consider the time complexities differently for different phases. If we consider the map phase, total work depends on three factors: first is the number of data points present in the data sets for which the computations are being performed, secondly the number of desired clusters and third the dimension of the data points. The number of data points along with their dimension decides the communication costs but we can definitely reduce it by using combiners. Also the reducer cost mainly depends on the number of data points only.

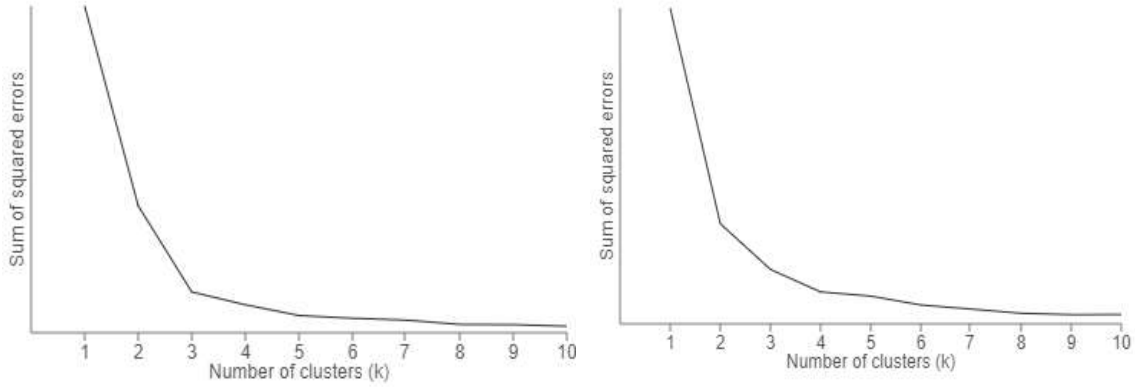
As we discussed earlier the biggest disadvantage of k-means is the selection of an optimum value of k that gives us the best desired results. The selection of a wrong value might take us to poor results. Also, in k-means it does not matter if the result is proper or not, it would just show us the clusters. Hence, we need a way to define and get to know whether the value of k that we are using is the right number of clusters or not. So let us discuss one of the methods to determine the correctness of the value of k. The method we are discussing is known as Elbow method.

The basis of elbow method is to execute k-means for a definite range of values for the data set where we want to partition the. So basically, we will compute the clusters using k-means for a certain range of values and then we have to calculate the SSE i.e. Sum of Squared Errors.

We have to calculate SSE for all the values of k for which we are computing the clusters. Then we can construct the graph between the values of k and the corresponding SSE value.

The graph is a bit arm shape and the nest value that can be considered for k is where the curve bends like elbow and thus gets its name- Elbow curve.

The curve clearly shows that as the value of k increases the value of SSE keeps on decreasing and when the value of k approaches n i.e. the number of data points within the data sets then, the value of SSE approaches 0. That would be best ideally, but we need lesser number of clusters such that we even get a lower SSE. So the idea behind this elbow curve is that the beginning of elbow demarcates the point from where further decrement in the value of k would lead to diminishing returns in SSE as shown in Figure2.7(a).



(a) (b)

Figure2.7: (a) and (b) represents the Elbow curve [14]

But, there might be the cases where our working and identification with the elbow curve might also fail as sometimes the elbow is not clearly visible in the plot as shown in Figure2.7 (b). [14] There might be the cases when we won't even get the arm like structure instead, we would get smooth curves and thus we need other methods for computing the nest desired value for k in the application of k-means algorithm

2.6. K-Means++

Another algorithm that can be used for partitioning of data sets is the extension of standard k-means only. The only difference between k-means and k-means++ is the way in which centroids or means are initialized in the very first iteration. In k-means, *k* centroids are

initialized randomly where as in k-means++ there is defined algorithm according to which the centroids are initialized and the rest of the iterations are same in both of them. Basically, this algorithm helps us to define the centroids that are well separated from each other. This approach was developed to improve the total error that is left in the end i.e. when we reach convergence condition.[20]

In the initialization process in this algorithm, the first mean is chosen at random out of k, so we have to select *k*-1 more means from the data points. Now the squared distance is calculated between data points and the nearest centroid. Then, we can choose the next mean whose probability is proportional to the squared distance i.e. the one that would be far would have more probability of being chosen. Thus we can repeat the above iteration again and again i.e. *k*-1 times so as to get *k* initial centroids that we require to begin with the standard k-means algorithm.[13][21]

If we consider the complexity of this initialization algorithm, it again depends upon the number of data points present in the data sets that we are considering , the dimensions of the data points and the number of means that have been initialized already. So, the number of means already initialized will keep on incrementing with every iteration. Thus, the complexity depends upon the square of number of clusters to be found and the number of data points to be performed computations upon and their dimensions as well.[22]

But, again here we have an issue in selecting the value of K. The selection of a wrong value might take us to poor results. Also, in k-means it does not matter if the result is proper or not, it would just show us the clusters. Hence, we need a way to define and get to know whether the value of k that we are using is the right number of clusters or not.

We can apply k-means++ also using a distributed setting. We can apply the distributed version using the map reduce paradigm. As we know the map reduce paradigm works in hadoop where we can split the computations into two parts: using map method and using reduce method. So basically, we will be breaking each iteration into two steps, first step would be applied using mapper function and second using reducer function.

Now in k-means++ we have two phases: first to calculate the squared distance between each point and its nearest centroid and second, to add new centroid to the set by selecting the one with the largest distance i.e. the largest probability. So, we can use map reduce to apply this algorithm with two different phases such that mapper function would operate on each data point in the dataset. The mapper function then computes the squared distance between the data point and each centroid and computes the nearest centroid to that data point on which it is being operated. Thus, here mapper function generates a single value only i.e. the squared distance from the nearest centroid. The reducer function takes this value as an input and it aggregates these emissions. Basically, the aggregation is done based on the probabilities i.e. the squared distances.

Thus, the map reduce produces a single value as an output i.e. the centroid that needs to be added to the set of centroids. As we discussed earlier, the new set of means has to be broadcasted to all the data sets.

If we consider the time complexity of this algorithm using map reduce it depends upon the current set of initialized means. Also we have the communication costs that depend upon the number of data points present in data sets and their dimension and the number of means or centroids to be chosen as well. Since, the reducer function that we are using for computation is also commutative as well as associative thus, we can reduce the communication costs also.

2.7. Choice of Python

For the project, python has been chosen as the programming language. Python is easy to learn and provide a number of built-in functions. Python modules help to make our work easier. Numpy module can be used to create random data set. It provides a number of built-in functions to compute mean, median etc. in minimum time. [23] Pandas module is a important module for data analysis. It provides a number of functionalities to work with the dataframes. It has become easy to search content, analyze data in the csv and text files with the help of Pandas.[23] Python also provides a built-in function to compute the k-clusters using k-means algorithm.[24] Matplotlib module is an important module for the project that helps us to

visualize the implementation. It is a module that helps to create plots and graphs. This module of python is of great help for this project.

Since Python provides a handful of useful libraries that can make our work easier, so Python is chosen to implement the algorithms.[25]

CHAPTER 3

SYSTEM DEVELOPMENT

In this chapter, the machine configurations, softwares used, technologies used have been discussed. Also, the algorithms used have been discussed in this.

3.1. System Design

In this section, the machine configuration and the softwares used to implement the map reduce clustering algorithms have been discussed. Machine configuration plays an important part in the working and efficiency of the algorithm. Moreover, the efficiency also depends on the versions of software used and the technologies implemented.

3.1.1. Hardware Requirements

To implement the traditional k-means, k-means via map reduce and the modified MapReduce algorithm, a machine is needed. The hardware specifications of the machine used for implementing the project are as follows:

- Processor: Intel(R) Core(TM) i5-5200U CPU @2.20GHz 2.20GHz
- RAM: 8GB
- 64-bit Operating System(CentOS 7)

3.1.2. Software Requirements

For this project, different data sets of different sizes are taken into consideration. To manage different sizes of data efficiently, Python is chosen as the programming language because python provides pandas module that makes data analysis a lot much easier. Thus, Spyder needs to be installed. Moreover, for running the Map Reduce task, Apache Hadoop needs to be installed. Thus, the softwares used for implementing the project are as follows:

- Apache Hadoop 2.7.4
- Spyder

3.1.3. Technologies used

In the project, the main objective is to form clusters. Thus, data clustering and data partitioning are the important aspects of this project. Moreover, when mapper and reducer functions are written in Python, then Hadoop Streaming helps to run our mapper and reducer in Apache Hadoop. Thus, the important aspects needed in this project are:

- Data Clustering
- Data Partitioning
- Hadoop Streaming

3.2. k-means Algorithm

Let X={x_1, ……, x_n} be a set of n data points, each with a dimension d. The objective of k-means is to find a set of k means M = {m_1, …., m_k} which minimizes the function

$$f(M) = \sum_{x \in X} \min_{\mu \in M} ||x - \mu||_2^2$$

In other words, k-means aims to minimize the Euclidian distance between every data point and the mean closest to that point. It is a NP-hard problem. So, exact solution may or may not exist. There are other good algorithms that gives approximate solutions for the problem.

3.3. Sequential Clustering

To study how effective MapReduce clustering algorithms are when compared to the sequential clustering algorithms, we perform serial k-means algorithm which consists of three phases:[16]

1. Startup : *k*-centroids are initialised randomly and it checks for the convergence criterion
2. Assignment: Each data point is assigned to one of the *k* centroids or cluster
3. Update: The centroid of the cluster is calculated again

3.3.1. Start-up

In the start-up program, we initialise *k* centroids randomly from the set of *n* data points. Then, we call the *Assignment* procedure where each data point is assigned to the cluster id which contains the centroid with the minimum distance. The centroids are calculated again in the *Update* procedure. The Start-up program, then, repeats the *Assignment* and *Update* procedure until the convergence criterion is met or the position of centroids remains the same.

Algorithm 1: Algorithm for Startup

Require:

- A set of *n* data points data points X= {x_1, x_2,...., x_n} of dimension *d*
- *k*, which specifies the number of final clusters to be formed where $k < n$

Output:

- A new set of centroid,
- number of iterations used to output the final clusters,
- *k* number of clusters and
- time taken for convergence

1: *centroid_list1* <= randomly choose *k* centroids
2: **initialise** *convergenceTime*, *Iterations*, *finalCluster*
3: *startTime* <= *currentTime()*
4: *M* <= call **Assignment**
5: *updated_centroid* <= *M*
6: *Iterations* <= 1
7: **while** (*updated_centroid* != *centroid_list1*) || (threshold limit is met) **do**
8: *centroid_list1* <= *updated_centroid*
9: *M* <= call **Assignment**
10: *Iterations* <= *Iterations* + 1
11: *updated_centroid* <= *M*
12: **end while**

13: *endTime* <= *currentTime()*

14: *convergenceTime* <= *(endTime - startTime)*

16: *finalClusters* <= *updated_centroids*

17: **write** *finalClusters*

18: **return** *finalClusters*, *Iterations* , *convergenceTime*

3.3.2. Assignment

In this phase, we use Euclidean distance to calculate the distances. For each data point, the distance is calculated between the data point and the centroid of every *k*- cluster. The centroid which gives the minimum distance with the data point is noted and the data point is assigned to the cluster containing that centroid.

Algorithm 2: Algorithm for Assignment

Require:

- A set of *n* data points data points X= $\{x_1, x_2, ..., x_n\}$, each of dimension *d*
- initial list containing *k* centroids C = $\{c_1, c_2, ..., c_k\}$

Output: A dictionary list consisting of each centroid and the data points assigned to them. This list is then passed as an argument to the CentroidCalculator program.

1: **Initialize** *clusters* as dictionary

2: *centroid_set* <= *C*

3: distance (x, y) = $\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$

where x_i (or y_i) is the coordinate of x (or y) in dimension i

4: **for all** $x_j \in X$ such that 1 <= j <= *n* **do**

5: *chosenCentroid* <= null

6: *minDistance* <= ∞

7: **for all** *centroid* **in** *centroid_set* **do**

8: *dist* <= *distance*(x_j, *centroid)*

9: **if** *chosenCentroid = null || dist < minDistance* **then**

10: minDistance <= dist

11: chosenCentroid <= centroid

12: **end if**

13: **end for**

14: clusters[*chosenCentroid]* <= *(x_j)*

15: j +=1

16: **end for**

17: call **CentroidUpdate**(*clusters*)

3.3.3. Centroid Update

The dictionary list that contains the centroids and the list of data points assigned to each centroid is passed as an input to the **CentroidUpdate** procedure. For each centroid, it loops through the data points assigned to it and calculate the new mean which becomes our updated centroid of the cluster. It, then, outputs the list of new centroids of k clusters.

Algorithm 3: Algorithm for CentroidUpdate

Require:

Input: List of centroids and the list of data points assigned to these centroids

Output: List of updated centroids

1: *output* <= Output from **Assignment**

2: *a* <= *output*

3: *updatedCentroidList* <= null

4: **for all** *c* **in** *a* **do**

5: *updatedCentroid, sum, numofPoints*<= null

6: **for all** *point* **in** *a[c]* **do**

7: *sum* += *point*

8: *numofPoints* += 1

9: **end for**

10: *updatedCentroid <= (sum / numofPoints)*

11: *updatedCentroidList***.append** *(updatedCentroid)*

12: **end for**

13: **return** *updatedCentroidList*

3.4. k-means++

k-means++ is similar to k-means. It uses the same iterative process of k-means but uses a different initialization approach. It chooses the initial means in a different manner. The initial means are chosen such that they are far apart from each other.

Algorithm 4: Algorithm for Initialization of k-means++

1: Choose a data point randomly from the set of n data points

2: For each point, compute the distance d(x) between the data point and the centroid (that has already been chosen)closest to that point.

3: Choose the next centroid such that the probability is proportional to $d(x)^2$

4: Repeat steps 2 and 3 until we get the k- centroids

3.5. k-means via MapReduce

K-means is a clustering algorithm used to cluster a set of data objects into k number of clusters based on the Euclidean distance. The first step is keep the data points organized and generate the initial centroids using the initialization procedure of k-means++ and write these to the initial centroid file.

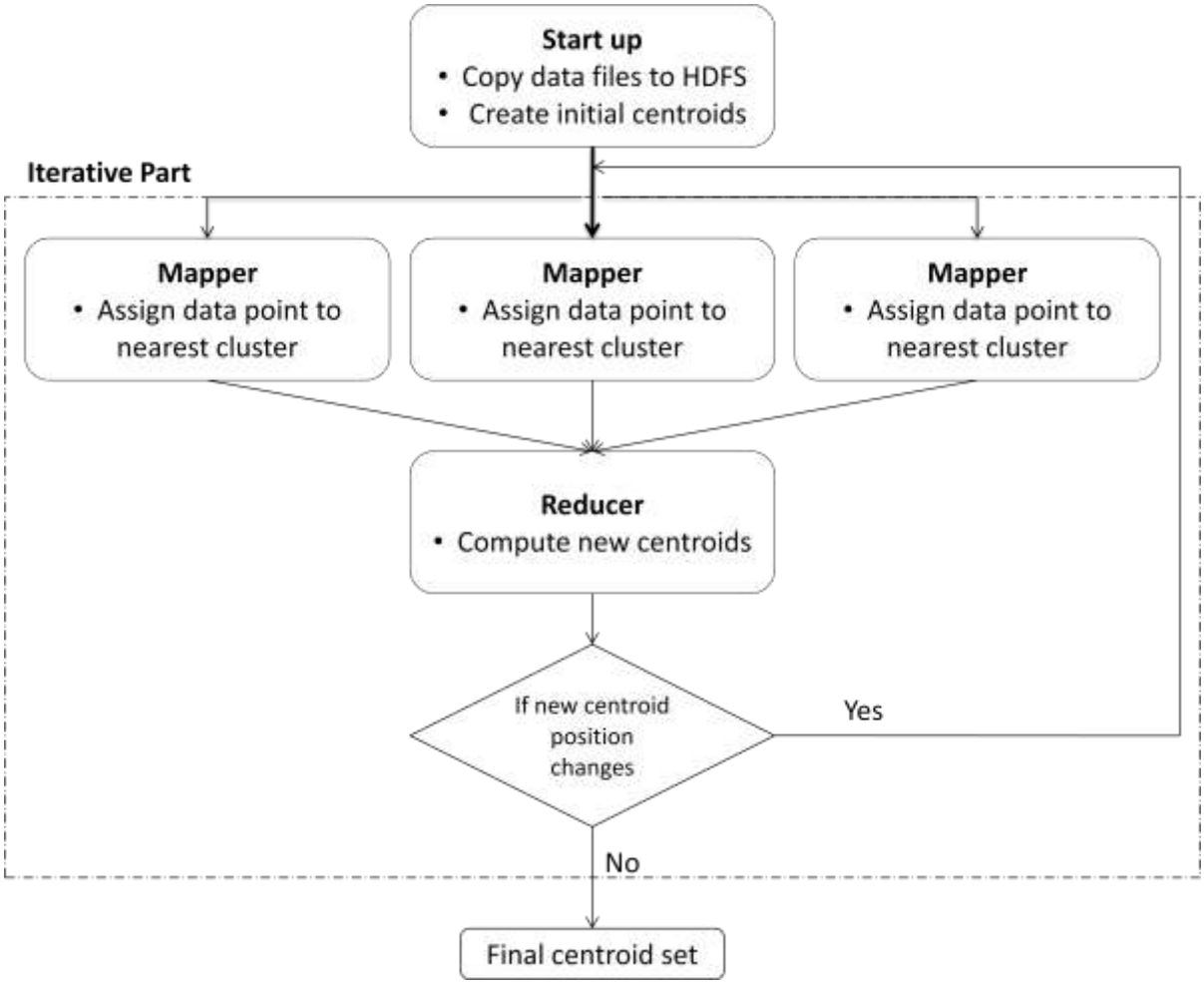


Figure 3.1. Structure of k-means via MapReduce algorithm

3.5.1. Start up

The start-up procedure is used to initiate the process and for the initialization of centroids of k clusters. It, then, calls the MapReduce procedure. In the mapper, each data point is assigned to the centroid(or cluster bearing the centroid) closest to the data point. The mapper function then calls the reducer. In the reducer, the new centroid is calculated by taking the mean of the data points associated with the cluster.

The start-up procedure is also responsible for checking the convergence condition in which it checks whether the position of new centroids is different from that of current centroids or not. If different, then the MapReduce procedure is executed again. Otherwise, the new centroids and final clusters are written to a file.

Algorithm 5: Algorithm for Startup

Require:

- A set of *n* data points X= {x_1, x_2,...., x_n}, each of dimension *d*
- *k* that specifies the number of clusters where $k < n$
- initial list of centroids $C = \{c_1, c_2, ..., c_k\}$ contained in the initial centroid file

Output: a new set of centroids, final clusters

1: *centroids_list1* <= *C*

2: **Initialize** *convergenceTime, Iterations, finalClusters*

3: *startTime* <= *currentTime()*

4: *M* <= perform **MapReduce** using Hadoop Streaming

5: *updated_centroidList* <= *M*

6: *Iterations* <= 1

7: **while** (*updated_centroidList* != *centroids_list1*) || (threshold limit is met) **do**

8: *current_centroids* <= *updated_centroidList*

9: *C*' <= perform **MapReduce** using Hadoop Streaming

10: *Iterations* <= *Iterations* + 1

11: *updated_centroidList* <= *M*

12: **end while**

13: *endTime* <= *currentTime()*

14: *convergenceTime* <= *(endTime - startTime)*

16: *finalClusters* <= *updated_centroids*

17: **write** *finalClusters*

18: **return** *finalClusters*, *Iterations* , *convergenceTime*

3.5.2. Mapper

The input data files is distributed to multiple mapper. The number of mappers assigned to do the task depends on the data size. We can not specify the number of mappers for the job. This task of choosing the number of mapper is done by Hadoop streaming. The file containing the

centroid of k-clusters is stored in a common directory accessible to all mapper or is distributed to each mapper separately.

The centroid list contains the cluster id to which it belongs and the cluster id works as the key and the centroid is chosen as the value. Each input data point in the subset (x_1, x_2, ..., x_m) is assigned to the closest centroid by the mapper procedure. Euclidean distance is used to assign the data point to the closest mean. When all the data points are assigned to the centroids by the mapper then Hadoop streaming job itself manages the task of sorting <key, value> pairs outputted by the mapper and arranges the output in the form <key, list of values>.

Algorithm 6: Algorithm for Mapper

Require:

- A subset of data points, $\{x_1, x_2, ..., x_n\}$ which is passed to every mapper
- initial list of centroids C = $\{c_1, c_2, ..., c_k\}$

Output: A list of centroids and data points assigned to each centroid. This dictionary list is written down to file and passed to the reducer.

1: X_{subset} <= $\{x_1, x_2, ..., x_m\}$

2: *centroids_set* <= *C*

3: distance (x, y) = $\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$

where x_i (or y_i) is the coordinate of x (or y) in dimension i

4: **for all** $x_j \in X$ such that 1 <= j <= m **do**

5: *chosenCentroid* <= *null*

6: *minDist* <= ∞

7: **for all** *cent* **in** *centroids_set* **do**

8: *dist* <= *distance(x_j, cent)*

9: **if** *chosenCentroid = null || dist < minDistance* **then**

10: minDistance <= dist

11: chosenCentroid <= cent

12: **end if**

13: **end for**

14: **Append** (x_i, *chosenCentroid*) to *outputList*

15: i +=1

16: **end for**

17: **return** *outputMapper*

3.5.3. Reducer

The output of the mapper is fed in as input to the reducer. It accepts the <key,value> pair output from the mapper. For each centroid, the reducer calculates a new value based on the list of data points passed along with the centroid. This updated centroid list is emitted as the output of the reducer which is sent back to the start-up program to check for convergence.

Algorithm 7: Algorithm for Reducer

Input: <key, value> where key = *chosenCentroid* (*cluster_id*) and value = list of data points assigned to the *cluster_id*

Output: <key, value> where key = *cluster_id* and value = *updatedCentroid*, which is the new centroid value calculated for the *cluster_id*.

1: *output* = Output from **Mapper**

2: *a* = {}

3: *updatedCentroidList* <= null

4: **for all** z **in** *output* **do**

5: *cluster_id* <= z.*key*

6: *data* <= z.*value*

7: *a[cluster_id]* <= *data*

8: **end for**

9: **for all** *cluster* **in** *a* **do**

10: *updatedCentroid, sum, numofPoints*<= null

11: **for all** *point* **in** *a[cluster]* **do**

12: *sum* += *point*

13: *numofPoints* += 1

14: **end for**

15: *updatedCentroid* <= *(sum / numofPoints)*

16: **Append** < *cluster*, *updatedCentroid* > **to** *updatedCentroidList*

17: **end for**

18: **return** *updatedCentroidList*

3.6. Modified MapReduce clustering algorithm

In k-means and k-means clustering algorithm, the assignment and update phase is carried out iteratively until convergence criteria is met. Since assignment phase involves calculating distance between data point and centroid, so the number of computations to calculate distance increases with the number of iterations. Moreover, the same computation may occur several times in the iterative process.

To minimize such number of computations, we can compute the pair- wise distance between the data points and store them in a comma separated file(csv file). Moreover, we modify our initialization procedure for initializing the centroids such that the initial centroids are a subset of the dataset.

The python scripts needed to implement this algorithm:

1. Initialization script
2. Distance computation script
3. Mapper script
4. Reducer script
5. Driver program script

The functioning of these scripts and the algorithms used are explained in further.

3.6.1. Initialization

In this, k- centroids are initialized using the k-means++ initialization algorithm.

Algorithm 8 : Initialization algorithm

```
1:  init_centroids <= null
2:  if length(init_centroids) = 0
3:      s <= choose x_i randomly from X
4:      Append s to init_centroids
5:  else if length(init_centroids) =1
6:      s <= init_centroids[0]
7:      k <= find x_j such that distance(x_j , s) is maximum
8:  else
9:  maxDist <= -1
10:  maxDistCentroid <= -1
11: for x_i in X
12:     closestCentroidDist <= ∞
13:     for c in init_centroids
14:         if i not in init_centroids
15:             if distance(x_j , c) < closestCentroidDist
16:                 closestCentroidDist <= distance(x_j , c)
17:             end if
18:         end if
19:     end for
20:     if square(closestCentroidDist) > maxDist and closestCentroidDist != ∞
21:         maxDist <= square(closestCentroidDist)
22:         maxDistCentroid <= x_i
23:     end if
24:     Append maxDistCentroid to init_centroids
25: end for
26: return init_centroids
```

Call this algorithm k- times so that we have k initial centroids.

3.6.2. Distance computation

In this, the pair-wise distance between the data points is calculated and the distance matrix is written down in a csv file. This csv file is stored at a common location and is accessible to all the mappers.

To compute the pair-wise distance of data points, we used the **scipy.spatial** module of Python. This module makes our task of finding pair-wise distance easy. For using the **distance_matrix()** function of scipy module, we need to first input the data points in the numpy array, *dist* and pass this array as an argument to the **distance_matrix()** function as shown below:

```
>>> scipy.spatial.distance_matrix(dist, dist)
```

This method works fine when we have numerical data. But this method doesn't work for the calculating the distance between the strings.

3.6.3. Mapper

In this, each data point belonging to the dataset is assigned to the closest centroid and distance between the data point and the centroid is read from the csv file containing the pair-wise distance matrix.

Algorithm 9 : Mapper algorithm

Input:

i. set of cluster centers(i.e. centroids)
ii. x_i data point present in X

Output $< z_i , x_i >$, **where** z_i is cluster id and x_i is the data point

1: **mapper (** ***cluster_center_set*** , $\boldsymbol{x_i}$){

2: *df* <= **load** *dataPointdistance.csv* using csv reader

3: *nearest_cluster_id* <= null

4: *nearest_distance* <= ∞

5: **for** *c* **in** *cluster_center_set*

6: *dist* <= *df* [x_i] [*c*]

6: **if** *dist* < *nearest_distance*

7: *nearest_distance* <= *dist*

8: *nearest_cluster_id* <= x_i

9: **end if**

10: **end for**

11: **emit** (*nearest_cluster_id* , x_i)

12: }

3.6.4. Reducer

In this, new centroids are calculated. Previously, in k-means we were using mean to find the new centroid. But in this modified MapReduce clustering algorithm, we've have used median to find the new centroid. The data point that has the median distance from the current centroid is chosen as the new centroid.

Algorithm 10: Reducer Algorithm

Input

i. *j*, cluster label key

ii. data points assigned to cluster *j*

Output: < *j* , *updatedCentroid* >

1: **reduce (** ***j*** , ***x_in_cluster_j*** [x_1, x_2, ...]){

2: *df* <= **load** *dataPointdistance.csv* using csv reader

3: **for** x_i **in** *x_in_cluster_j*

4: *dist* <= *df* [x_i] [*j*]

5: store the data point x_i associated with *dist*

6: **Append** *dist* **to** *distList*

7: **end for**

8: find the median distance present in *distList*

9: x_j <= data point associated with the median distance is the new centroid

10: **emit** (j , x_j)

11: }

3.6.5. Driver program

This script is responsible for running the above mentioned scripts. Firstly, the centroids are initialized and the pair-wise data-point distances are written down to csv file. Then, it is responsible for calling the mapper and reducer scripts iteratively using Hadoop Streaming. It calls the MapReduce functionality until the convergence criteria is met or the threshold number of iterations is reached. Convergence criteria is met when there is no change in the positions of current centroids and the new centroids.

Before running the driver script, the dataset is loaded on the HDFS.

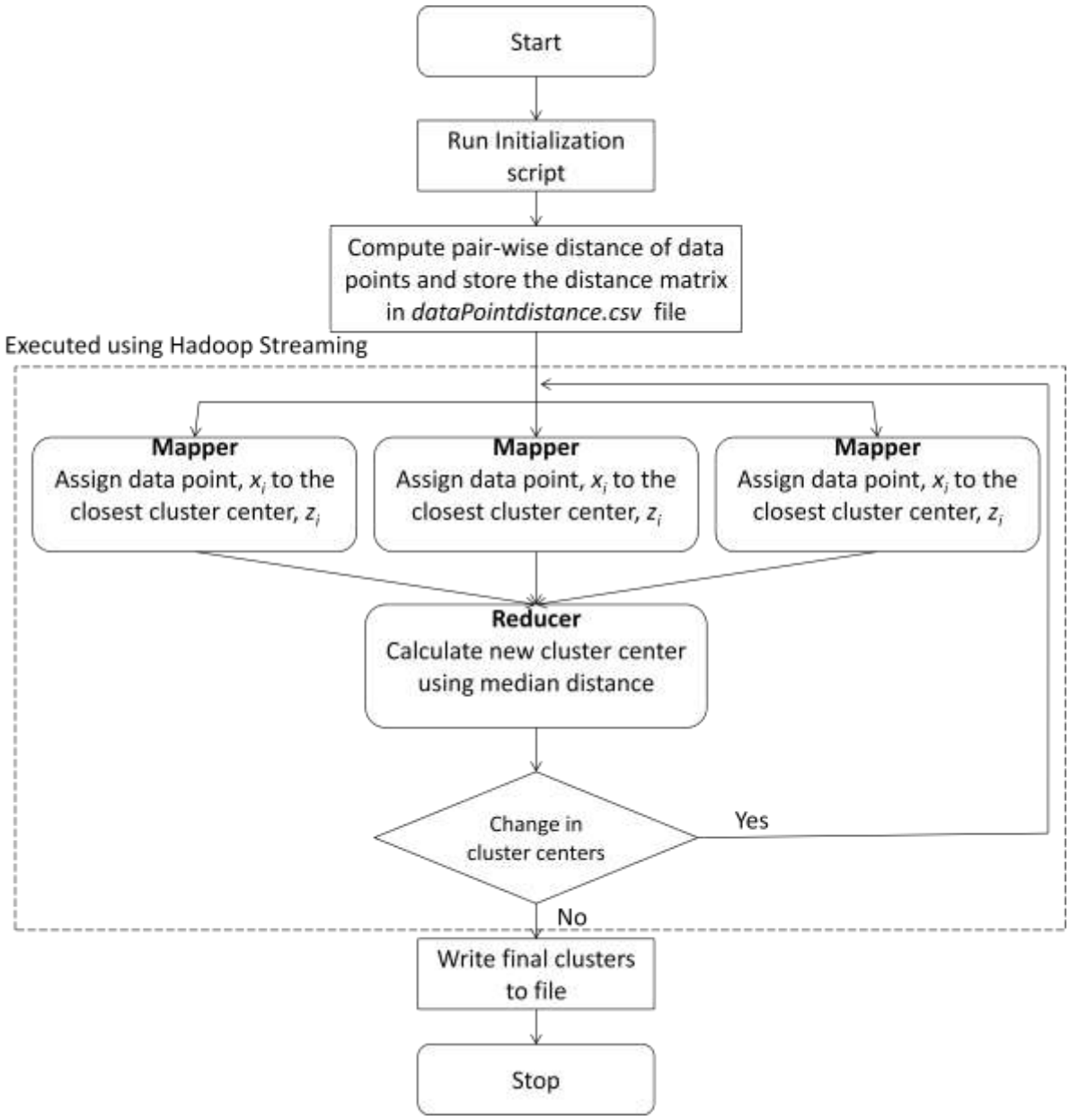


Figure 3.2. Structure of driver script of Proposed Algorithm

CHAPTER 4

PERFORMANCE ANALYSIS

In this chapter, two experiments which include two different datasets have been discussed in detail. The efficiency and performance comparison of k-means, k-means via Map Reduce and the modified Map Reduce clustering algorithm was, thus concluded based on the results of the experiments and the results have been discussed in detail.

4.1. Experiment and Results

In this section, two experiments have been discussed. Experiment1 involves the use of small dataset and Experiment 2 involves a comparatively large dataset. The number of iterations, the convergence time taken by k-means, k-means via map reduce and the modified Map Reduce clustering algorithm were noted to compare the performance of the algorithms discussed in chapter two.

4.1.1. Experiment 1: Comparison of Map Reduce algorithms with small dataset

The objective is to compare the performance of k-means, k-means via map reduce and modified Map Reduce clustering algorithm on a small randomly generated dataset.

In this study, dataset of 1126 data points was randomly generated using Python modules. k-means, k-means via MapReduce and the modified MapReduce clustering algorithm is applied on this small dataset. After implementing these algorithms, the clusters were graphed to see if these algorithms form the same clusters or if clusters formed by these algorithms vary to a great extent.

These algorithms were applied to the dataset for k=3 and k=4.
The clusters formed after implementing the serial k-means for k=3 are shown in Figure4.1. In this figure, the blue data points plotted belong to cluster 1, the blue data points belong to cluster 2 and the green data points belong to cluster 3.

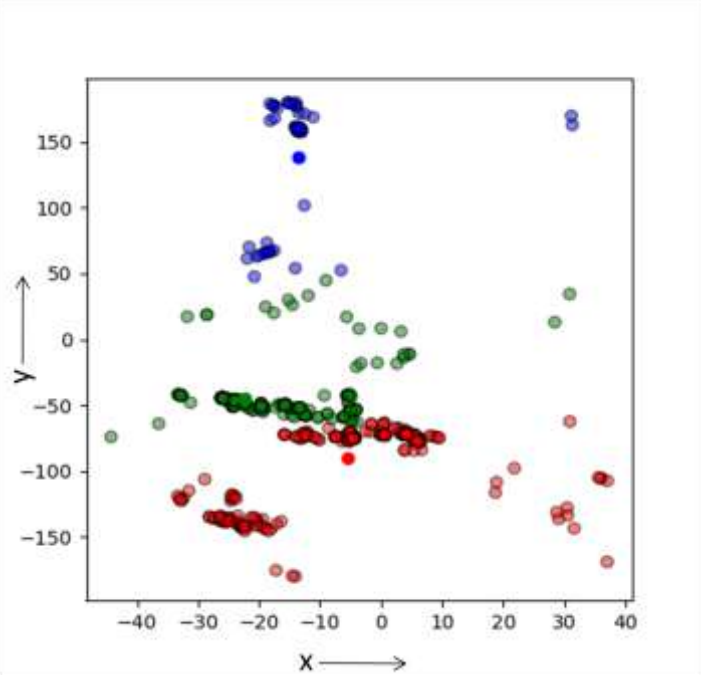


Figure4.1: Clusters formation using k-means clustering algorithm for k=3

The clusters formed after implementing the k-means via Map Reduce algorithm are shown in Figure4.2. In this figure, the green data points plotted belong to cluster 1, the orange data points belong to cluster 2 and the blue data points belong to cluster 3.

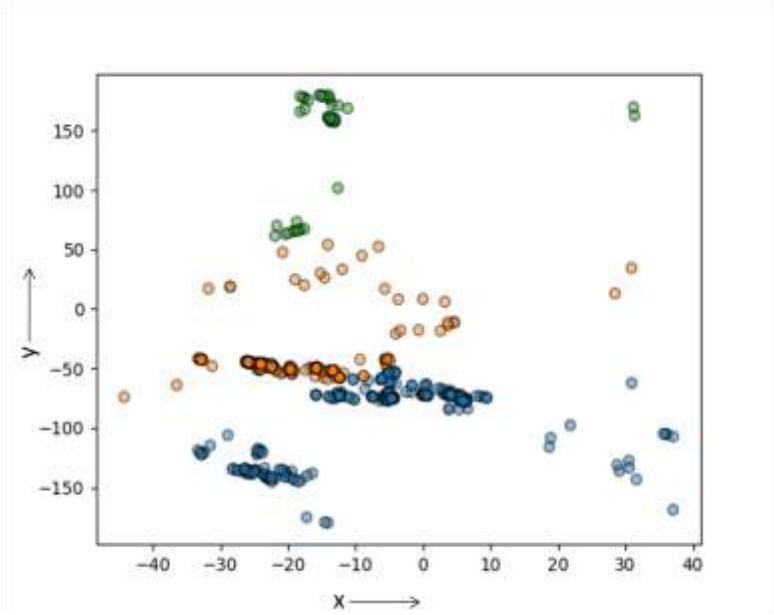


Figure4.2: Clusters formation using k-means via map reduce for k=3

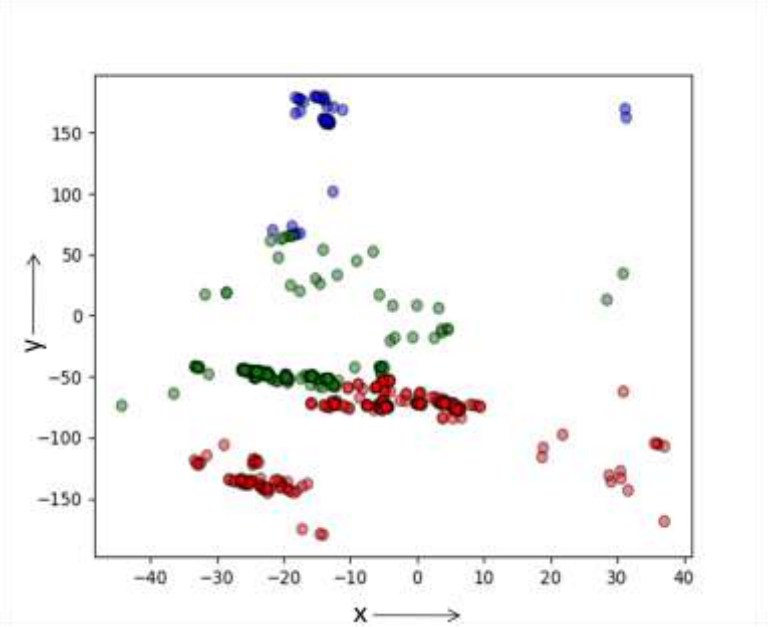


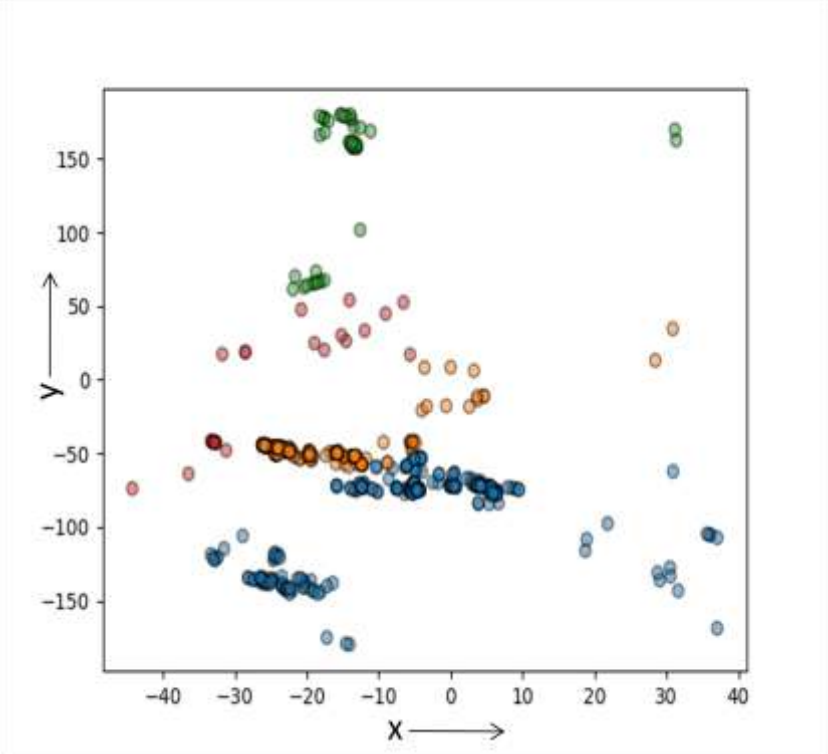
Figure4.3: Clusters formation using modified map reduce clustering algorithm for k=3

The three clusters formed after implementing modified Map Reduce clustering algorithm are shown in Figure4.3. In this figure, the blue data points plotted belong to cluster 1, the green data points belong to cluster 2 and the red data points belong to cluster 3.

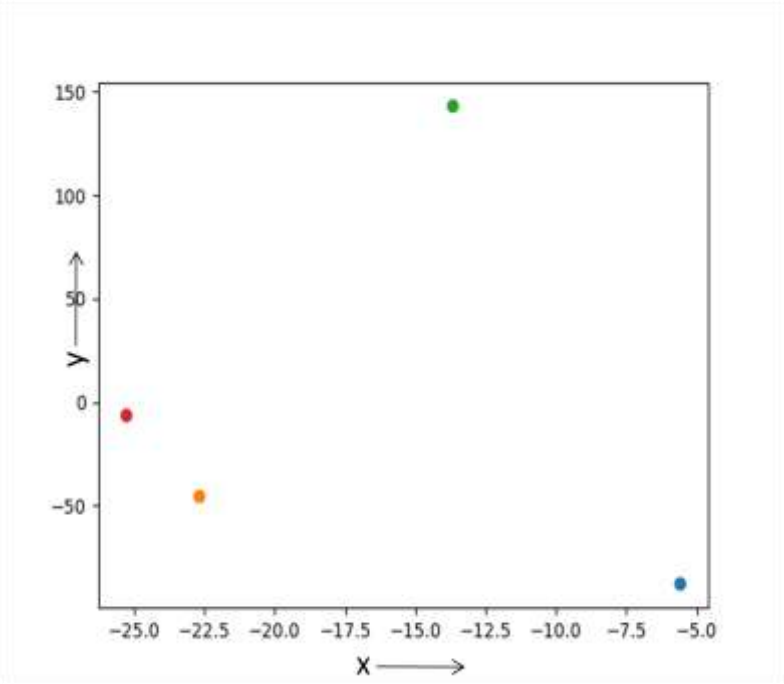
Figure 4.1, 4.2, 4.3 represents the clusters formed after applying different algorithms to randomly generated data set. From these figures, we observe that the clusters formed by these algorithms are similar. This similarity is the proof of accuracy. This means that the modified clustering algorithm is correct and produces the needed result.

We implemented the same algorithms to form 4 clusters(k=4). The clusters formed for k=4 is shown in the Figure4.4(a). The centroids of these four clusters are also plotted and are shown in Figure4.4(b).

In Figure4.4(a), the green data points belong to cluster 1, the red data points belong to cluster 2, the orange data points belong to cluster 3 and the blue data points belong to cluster 4.



(a)



(b)

Figure4.4: (a) shows the clusters formed and (b) shows the final coordinates for k=4

The time taken by these algorithms was thus noted to compare the performance and is listed as follows:

Algorithm implemented	Number of clusters (k)	Number of Iterations (i)	Time Taken (in seconds)
Serial k-means	3	5	0.08754
	4	3	0.25349
k-means via Map Reduce	3	5	131.8220
	4	3	69.12158
Modified Map Reduce clustering algorithm	3	5	128.7115
	4	3	68.2985

Table4.1: Comparison of the clustering algorithms

From the Table4.1, we plotted the graph.

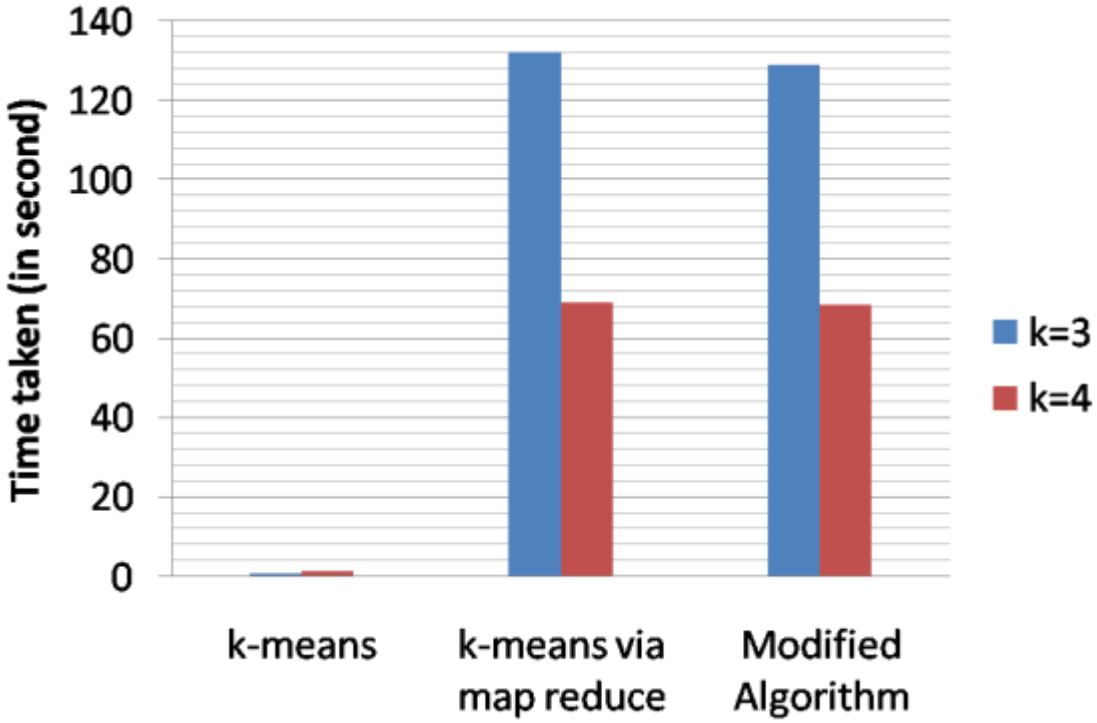


Figure4.5: Comparison of clustering algorithms for Experiment 1

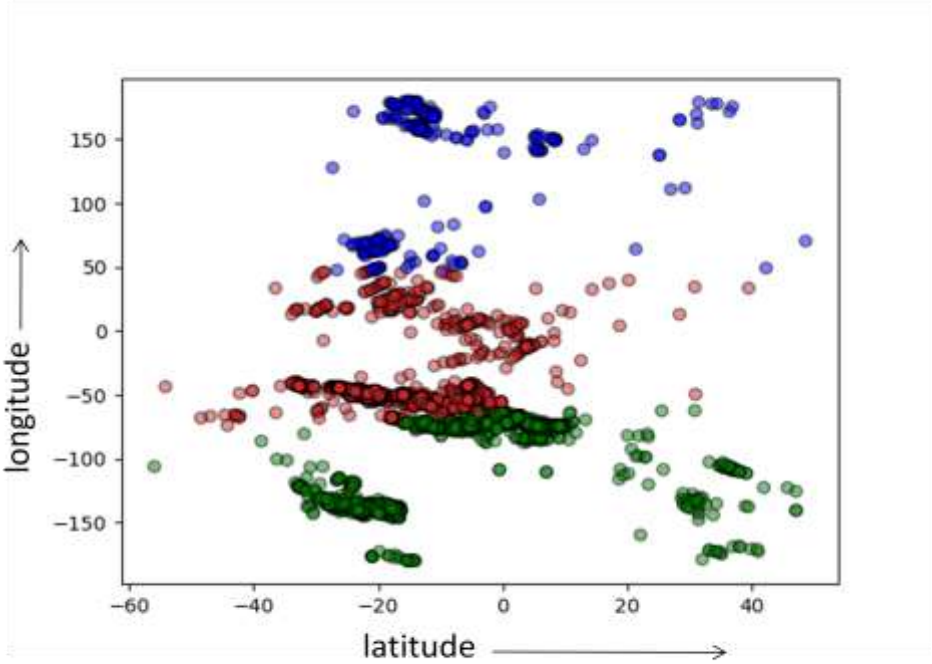
From the above plot, we observe that serial k-means is very fast as compared to other algorithms. The reason for the fast performance of serial k-means is attributed to the size of the data. The dataset is very small and thus, the map reduce algorithms doesn't show any performance improvement for such data. The time taken to set up the mapper and reducer is included in the total time and the time taken to exchange data between the mapper and the reducer adds to the time complexity for such small data in map reduce algorithms.

Another important thing to be noted in this plot is that the modified Map Reduce algorithm takes less time as compared to the k-means via map reduce. The reason behind this is the number of computations required to calculate the Euclidean distance in the mapper function. In the modified algorithm, we had already stored the pair-wise distances between the data points in a matrix and the distance matrix is stored in a common location from where all the mappers can access it. Hence, while implementing the modified algorithm, the distance matrix was directly used instead of computing the distance. this saved time.

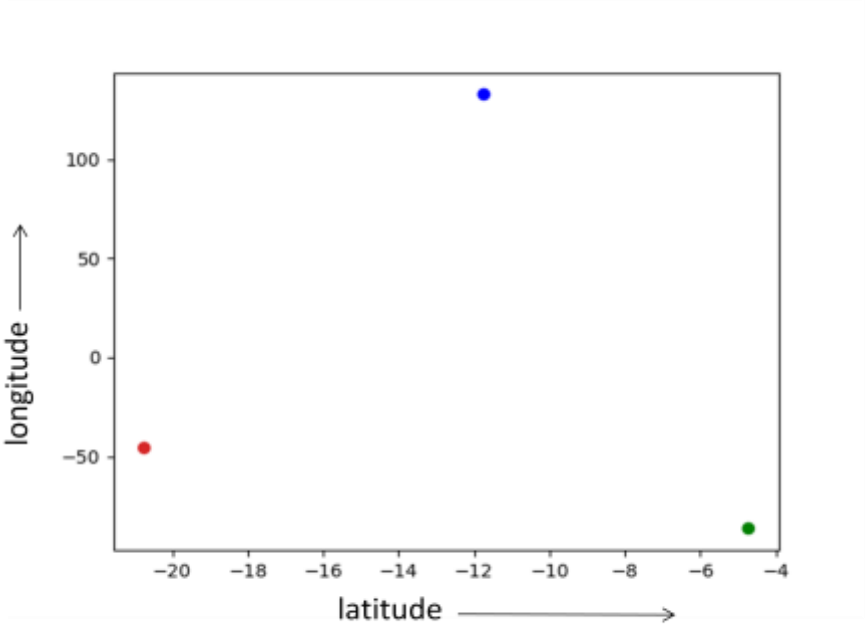
4.1.2. Experiment 2: Comparison of Map Reduce algorithms with large dataset

The objective of this experiment is to compare the performance of k-means via map reduce and modified Map Reduce clustering algorithm on a real time data taken from STARNET lightning network.

In this experiment, a subset of the STARNET lightning data is used consisting of 10788 data points. This dataset contains 29 attributes and contains no unique identifier for the data points. Thus, a unique identifier was added to uniquely identify the data points. Two attributes, the latitude and longitude, were used to cluster the data. k-means via map reduce and modified Map Reduce algorithm were implemented to see the improvement in the modified algorithm.



(a)



(b)

Figure4.6: (a) shows the clusters formed and (b) shows the final coordinates for k=3

To compare the performance of the two algorithms over the STARNET dataset, the time taken to form the final clusters was noted and is listed in the Table4.2.

Algorithm implemented	Number of clusters (k)	Number of Iterations (i)	Time Taken (in seconds)
k-means via Map Reduce	3	5	139.1514
	4	5	122.0934
Modified Map Reduce clustering algorithm	3	5	134.7864
	4	5	119.5520

Table4.2: Comparison of clustering algorithms for STARNET dataset

From this table, the bar chart has been plotted and is given in Figure4.7.

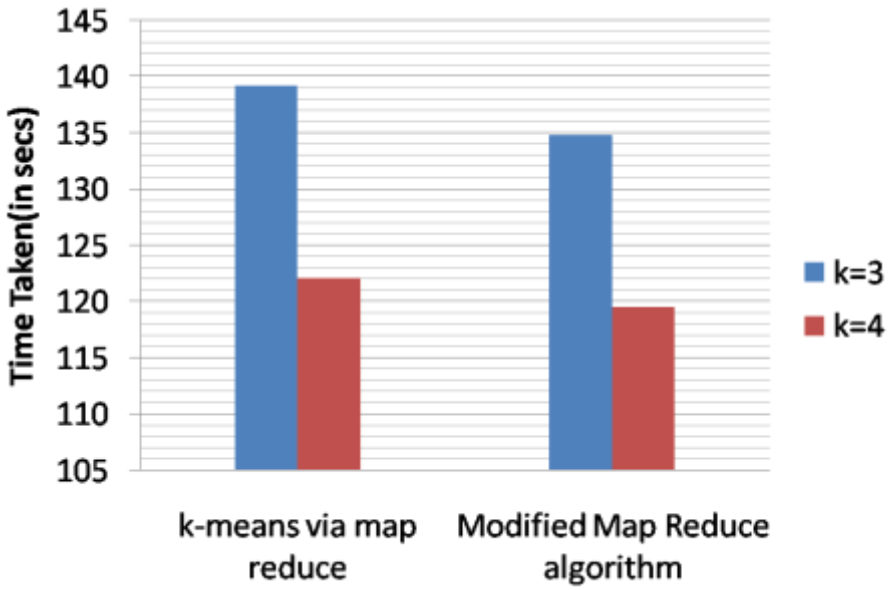


Figure4.7: Comparison of algorithms of Experiment 2

From this table, it is clear that the modified Map Reduce clustering algorithm takes less time as compared to the traditional k-means via map reduce. The performance of the modified clustering algorithm is better

CHAPTER 5

CONCLUSION

5.1. Conclusion

In this study, we have discussed the clustering algorithms used to partition the large datasets. k-means is a clustering algorithm in which initial centroids are selected at random. The random initialization of the centroids lead to varying number of iterations needed to cluster the dataset. Moreover, it is difficult to find the k- parameter that tells us about the number of clusters that can partition our data efficiently for further analysis. k-means++ is an advancement over k-means where initialization of centroids follows a different approach and the initialized centroids are well separated from each other. Both k-means and k-means++ involve a lot of computations to calculate the Euclidian distance between data point and the centroids of cluster. Moreover computations were needed to find the mean of the cluster. The problem of increasing number of distance computations and time lag due to these computations is solved by the modified MapReduce clustering algorithm discussed in this study.

The above discussed algorithm reduces the time taken to partition the given dataset but the large amount of memory space is required to store the distance matrix. Thus, the space complexity of this algorithm increases.

Map reduce clustering algorithms works best for the large datasets. But it doesn't work well for the small data sets because if the data is small that can be fitted in just a single mapper, then the overhead of passing data between mapper and reducer and other overheads of intermediate read/write make it unsuitable for smaller data sets. Performance of Map Reduce clustering algorithms can be increased by using the combiner since it reduces the intermediate read/write. Serial k-means algorithm works best for the small data but takes a lot of time to process large data sets.

5.2. Future Scope

Clustering using K-means through MapReduce is a good technique to cluster huge data. Even the modified MapReduce clustering algorithm turned out to be a better algorithm but the space complexity of storing the distance matrix is huge. Moreover, the distance function used in this study is applicable only for the numerical values. This distance function can be improvised to work for both numerical and string data. Moreover, concepts of distributive computing can be applied so that the files stored in the common location can be accessed without any communication delay.

REFERENCES

[1] Phaneendra, S. Vikram, and E. Madhusudhan Reddy. "Big Data-solutions for RDBMS problems-A survey." *12th IEEE/IFIP Network Operations & Management Symposium (NOMS 2010)(Osaka, Japan, Apr 19 {23 2013)*. 2013.

[2] Mythili, S., and E. Madhiya. "An analysis on clustering algorithms in data mining." *International Journal of Computer Science and Mobile Computing* 3.1 (2014): 334-340.

[3] Morales,C.A.; Neves. J.R; Anselmo. E.M.; "SFERICS TIMING AND RANGING NETWORK - STARNET: EVALUATION OVER SOUTH AMERICA" XIV *International Conference on Atmospheric Electricity*, August 08-12, 2011, Rio de Janeiro, Brazil

[4] Mishra, Shweta, and Vivek Badhe. "Improved Map Reduce K Mean Clustering Algorithm for Hadoop Architecture." *International Journal Of Engineering And Computer Science*5.7 (2016).

[5] Bernice Purcell "The emergence of "big data" technology and analytics" *Journal of Technology Research* 2013.

[6] Bhosale, Harshawardhan S., and Devendra P. Gadekar. "A review paper on Big Data and Hadoop." *International Journal of Scientific and Research Publications* 4.10 (2014): 1-7.

[7] Sagar S. Lad, Naveen Kumar, Dr. S.D. Joshi. "Comparison Study on Hadoop HDFS with Lustre File System"- *International Journal of Scientific Engineering and Applied Science(IJSEAS)*- Volume-1, Issue-8, November 2015

[8] HadoopTutorials.co.in " Understand the internals of HDFS block read/writes " *http://hadooptutorials.co.in/tutorials/hadoop/internals-of-hdfs-file-read-operations.html*

[9] Dr. T. Suryakanthi and V.S.J.Pallapolu. "A Comparative Study on Performance of Hadoop File System with MapR File System to process Big Data Records"- *IJCSI International Journal of Computer Science Issues,* Volume 13, Issue 1, January 2016.

[10] Jeffery Dean and Sanjay Ghemawat "MapReduce: Simplified Data Processing on Large Clusters" *Google, Inc.*

[11] P.Sudha and Dr. R. Gunavathi "A Survey Paper on Map Reduce in Big Data"- *International Journal of Science and Research(IJSR)* Volume 5, Issue 9, September 2016

[12] Piyush Gupta, Pardeep Kumar, Girdhar Gopal "Sentiment Analysis on Hadoop with Hadoop Streaming"- *International Journal of Computer Applications(0975-8887)* Volume 121- No.11, July, 2015

[13] Max Bodoia. "MapReduce Algorithms for k-means Clustering "
Retrieved from Stanford University Repository

[14] Robert Gove's Block "Using the elbow method to determine the optimal number of clusters for k-means clustering" *https://bl.ocks.org/rpgove/0060ff3b656618e9136b*

[15] Grace Nila Ramamoorthy "K-means Clustering Using Hadoop MapReduce " Retrieved from the UCD School of Computer Science and Informatics repository(2011)

[16] Kanungo, Tapas, et al. "An efficient k-means clustering algorithm: Analysis and implementation." *IEEE transactions on pattern analysis and machine intelligence* 24.7 (2002)

[17] Raval, Unnati R., and Chaita Jani. "Implementing and Improvisation of K-means Clustering." *IJCSMC* 4.11 (2015)

[18] Ekanayake, Jaliya, et al. "Twister: a runtime for iterative mapreduce." *Proceedings of the 19th ACM international symposium on high performance distributed computing*. ACM, 2010.

[19] Cui, Xiaoli, et al. "Optimized big data K-means clustering using MapReduce." *The Journal of Supercomputing* 70.3 (2014)

[20] Kapoor, Akanksha, and Abhishek Singhal. "A comparative study of K-Means, K-Means++ and Fuzzy C-Means clustering algorithms." *Computational Intelligence & Communication Technology (CICT), 2017 3rd International Conference on*. IEEE, 2017.

[21] Bahmani, Bahman, et al. "Scalable k-means++." *Proceedings of the VLDB Endowment* 5.7 (2012)

[22] Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007.

[23] McKinney, Wes. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc.", 2012.

[24] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *Journal of machine learning research* 12.Oct (2011)

[25] Millman, K. Jarrod, and Michael Aivazis. "Python for scientists and engineers." *Computing in Science & Engineering* 13.2 (2011)