A MAP REDUCE APPROACH OF K-MEANS++ ALGORITHM WITH INITIAL EQUIDISTANT CENTERS

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Krittika Bhattacharyya

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

October 2015

Fargo, North Dakota

North Dakota State University
Graduate School

Title

A Map Reduce Approach Of K-Means++ Algorithm with Initial Equidistant Centers

By

Krittika Bhattacharyya

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Chair

Dr. Maria Alfonseca Cubero

Dr. Saeed Salem

Approved:

10/21/2015	Dr. Brian Slator
Date	Department Chair

ABSTRACT

Data clustering has been received considerable attention in many applications, such as data mining, document retrieval, image segmentation and pattern classification. The enlarging volumes of information emerging by the progress of technology, makes clustering of very large scale of data a challenging task. In order to deal with the problem, many researchers try to design efficient parallel clustering algorithms. In this paper, we propose a parallel *k*-means++ clustering algorithm based on MapReduce, which is simple like traditional K-means, yet more powerful because the initial centroid selection process is not random. It follows a formula to plot initial centroids at equal distance and then iterates repeatedly like k-means to converge and produce final cluster. This makes this algorithm faster and parallelizing makes it more scalable. The experimental results demonstrate that the proposed algorithm can scale well and efficiently process large datasets.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my adviser Dr. Simone Ludwig for introducing me to the world of big data processing and map-reduce technique. I would specially like to thank her for being patient to my basic questions and explaining things whenever I needed her help; it motivated me to explore this field even further.

I am grateful to my committee members Dr. Maria Alfonseca Cubero and Dr. Saeed Salem for their invaluable time.

I would like to thank my parents and my fiancée for always having trust in my ability and for encouraging me in every possible ways.

.

DEDICATION

I want to dedicate this paper to my parents.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Data analysis underlies many computing applications, either in a design phase or as part of their operations. Data analysis procedures can be bifurcated as either exploratory or confirmatory based on the availability of appropriate models for the data source, but a key element in both types of procedures is the grouping, or classification of measurements based on either (i) goodness-of-fit to a postulated model, or (ii) natural groupings (clustering) revealed through analysis. Clustering is the unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters) based on similarity. The clustering problem has been addressed in many contexts and by researchers in many disciplines; this reflects its broad appeal and usefulness as one of the steps in exploratory data analysis.

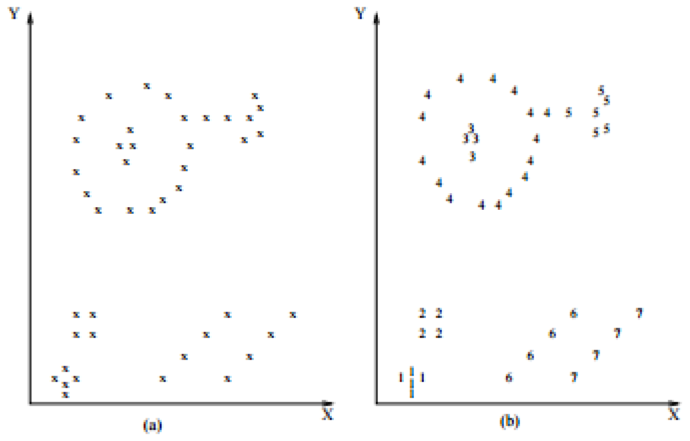

Figure 1. Data Clustering [1]

Cluster analysis itself is not one specific algorithm, but the general task to be solved. It can be achieved by various algorithms that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Popular notions of clusters include groups with small distances among the cluster members. Intuitively, patterns within a valid cluster are more similar to each other than they are to a pattern belonging to a different cluster. An example of clustering is depicted in Figure 1. [1] The input patterns are shown in Figure 1(a), and the desired clusters are shown in Figure 1(b). Clustering is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including learning, pattern, image analysis, information retrieval, and bioinformatics. [2]

However, with increasing volume of data and its heterogeneous nature, the clustering process is becoming more complex day by day. This paper concentrates on analyzing high volume of data with an optimized clustering technique.

1.1. Big Data

With the development of information technology, data volumes processed by many applications will routinely cross the peta-scale threshold – huge volume of data and that is 'big data'. However, volume of data is one of the attributes which makes a data set big data but it is not the only one. There are many other qualities, which makes big data different from traditional data set. [3] Some of them are:

Velocity of data: The data reception rate is much faster than traditional data. Big data velocity deals with the pace at which data flows in from sources like business processes, machines, networks and human interaction with things like social media sites, mobile devices, etc. The flow of data is massive and continuous. To cope up with that speed the processing speed also needs to be very fast.

Variety of data: There is no predefined structure for big data set. Rather, to be very specific, most of time it is semi-structured or unstructured. Previously we used to store data from sources like spreadsheets and databases. Now data comes in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. This variety of unstructured data creates problems for storage, mining and analyzing data. So, none of the traditional structured query language or processes works on that.

Values of data: A big data set has its own intrinsic values, which refers to the biases, noise and abnormality in data. Is the data that is being stored, and mined meaningful to the problem being analyzed? Values of big data is the biggest challenge when compared to things like volume and velocity. As part of the big data strategy data needs to be cleaned and processed to keep 'dirty data' from accumulating in systems.

Martin Wattenberg, Fernanda Viégas and, Katherine Hollenbach were able to present a visual representation of big data's properties using daily Wikipedia edits. [4] Figure 2 represents the visual effect.

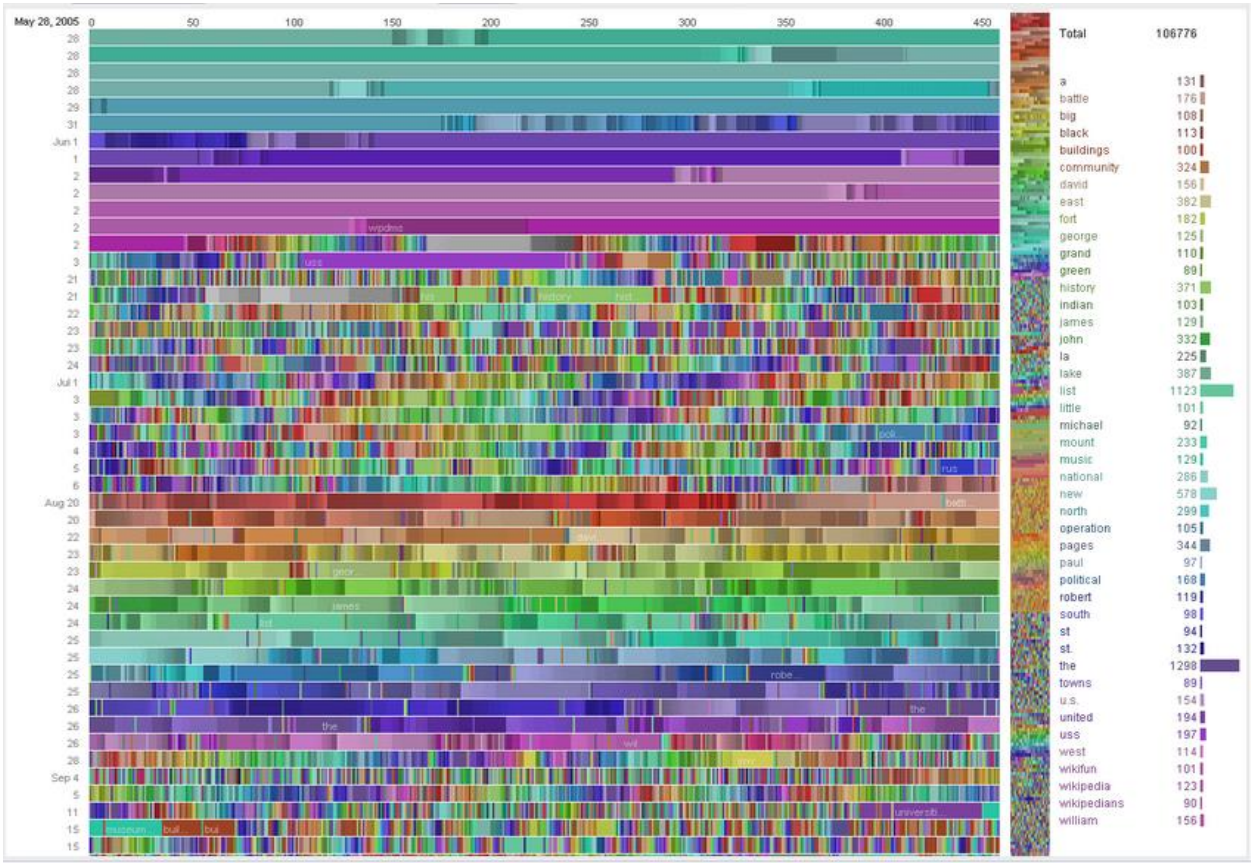

Figure 2. Visualization of Daily Wikipedia Edits Created By IBM [4]

Now when we know what does big data mean, the obvious next question is how to make it comprehensible? Big data sets are so large or complex that traditional data processing applications are inadequate. It is neither feasible to label this large collection of data nor can we have prior knowledge about the number and nature. Big data requires exceptional technologies to efficiently process large quantities of data within tolerable execution times. A 2011 McKinsey report [5] suggests suitable technologies include A/B testing, crowdsourcing, data fusion and integration, genetic algorithms, machine learning, natural language processing, signal processing, simulation, time series analysis and visualization. Among all these, clustering is one of the most accepted machine learning techniques which provides efficient browsing, search, recommendation and organization of data. Clustering can be defined as the key to big data problem because it can work with huge volumes of unstructured data (which is the main bottleneck for big data) [3]. Clustering can retrieve relevant information from unstructured data by categorizing it based on similarity.

1.2. Clustering Big Data

Clustering algorithms have emerged as an alternative powerful meta-learning tool to accurately analyze the massive volume of data generated by modern applications. In particular, their main goal is to categorize data into clusters such that objects are grouped in the same cluster when they are similar according to specific metrics. There is a vast body of knowledge in the area of clustering and there have been attempts to analyze and categorize them for a larger number of applications. However, one of the major issues in using clustering algorithms for big data that causes confusion amongst practitioners is the lack of consensus in the definition of their properties as well as a lack of formal categorization. [2] [6]

So far, several researchers have proposed some parallel clustering algorithms, which makes the big data analysis process even faster and scalable. But all these parallel clustering algorithms have the following drawbacks: a) They assume that all objects can reside in main memory at the same time; b) Their parallel systems have provided restricted programming models and used the restrictions to parallelize the computation automatically. Both assumptions are prohibitive for very large datasets with millions of objects. MapReduce is a concept or programming model, which can take care of this problem.

1.3. Map-Reduce

MapReduce is a programming model and an associated implementation for processing and generating large data sets. [1] [7]. The Map and Reduce functions of MapReduce are both defined with respect to data structured as (key, value) pairs. Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain. Figure 3 describes the overview of the Map function.

```
Map (k1, v1) → list (k2, v2)
```

Figure 3. Map Function

The Map function is applied in parallel to every pair in the input dataset. This produces a list of pairs for each call. After that, the MapReduce framework collects all pairs with the same key from all lists and groups them together, creating one group for each key.

The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same domain. Figure 4 represents the concept of the Reduce function.

```
Reduce (k2, list (v2)) → list (v3)
```

Figure 4. Reduce Function

Each Reduce call typically produces either one value v3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list. Thus, the MapReduce framework transforms a list of (key, value) pairs into a list of values.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

In general, we use a framework or commodity software called Hadoop (Apache Hadoop) [8] which helps to easily implement program into map reduce.

1.3.1. Hadoop

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then the inputs to the reduce tasks. Typically, both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically, the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

Minimally, applications specify the input/output locations and supply map and reduce functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the job configuration. The Hadoop job client then submits the job (jar/executable, etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client. [9]

In this paper, we implement the k-means++ (with Initial Equidistant Centers) algorithm within the MapReduce framework using Hadoop [10] to make the clustering method applicable to large-scale data. By applying proper <key, value> pairs, the proposed algorithm can be executed in parallel effectively. We conduct comprehensive experiments to evaluate the proposed algorithm. The results demonstrate that our algorithm can effectively deal with large scale datasets.

2. BACKGROUND

Clustering is useful in several exploratory pattern-analysis, grouping, decision-making, and machine-learning situations, including data mining, document retrieval, image segmentation, and pattern classification.

Different approaches to clustering data can be described with the help of the hierarchy shown in Figure 5. [11]

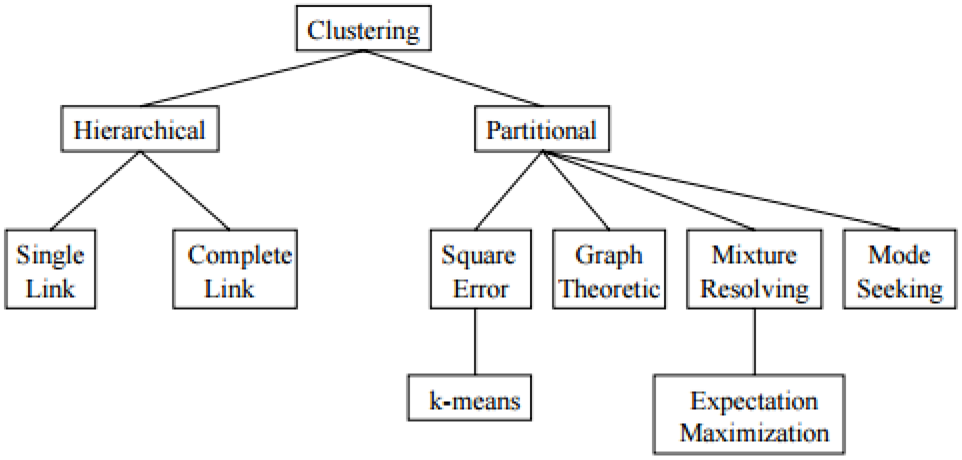

Figure 5. Taxonomy of Clustering Approach [11]

Among all, K-means is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The term "k-means" was first used by James MacQueen in 1967 [12], though the idea goes back to Hugo Steinhaus in 1957 [13]. The standard algorithm was first proposed by Stuart Lloyd in 1982 [14] as a technique for pulse-code modulation, though it was not published outside of Bell Labs until 1982 [15]. In 1965, Forgy published essentially the same method, which is why it is sometimes referred to as Lloyd-Forgy [16]. A more efficient version was proposed and published in FORTRAN by Hartigan and Wong in 1975/1979 [17] [18].

2.1. K-Means Algorithm

K-means clustering is a method of cluster analysis, which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. The most common algorithm uses an iterative refinement technique. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). The main idea is to define k centers, one for each cluster. These centers should be placed in an arbitrary way but intelligently because different initialization causes different results. So, the better choice of initialization makes the algorithm better. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is remaining, the first step is completed and the initial cluster set up is done. At this point we need to re-calculate k new centroids depending on the mean of the data of each group. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. This recalculation and association process iterates step by step until no more changes are done or in other words the centers do not move any more. Finally, this algorithm aims at minimizing an objective function know as squared error function given by [19]:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} (\|x_i - v_j\|)^2 \tag{1}$$

Where,

'$\|x_i - v_j\|$' is the Euclidean distance between x_i and v_j.

'c_i' is the number of data points in ith cluster.

'c' is the number of cluster centers.

The algorithmic steps for k-means clustering are as follows:

Let $X = \{x_1,x_2,..,x_n\}$ be the set of data points and $V = \{v_1,v_2,..,v_c\}$ be the set of centers.

Step 1) Randomly select 'c' cluster centers.

Step 2) Calculate the distance between each data point and cluster centers.

Step 3) Assign the data point to the cluster center whose distance from the cluster center is the minimum of all the cluster centers.

Step 4) Recalculate the new cluster center using:

$$v_i = (1/c_i) \sum_{j=1}^{c_i} x_i \tag{2}$$

Where, ‘c_i’ represents the number of data points in ith cluster.

Step 5) Recalculate the distance between each data point and new obtained cluster centers.

Step 6) If no data point was reassigned then stop, otherwise repeat from Step 3).

Figure 6 represents the overall K-Means algorithm concept, how it starts with random centroid selection and then iterates until it converges.

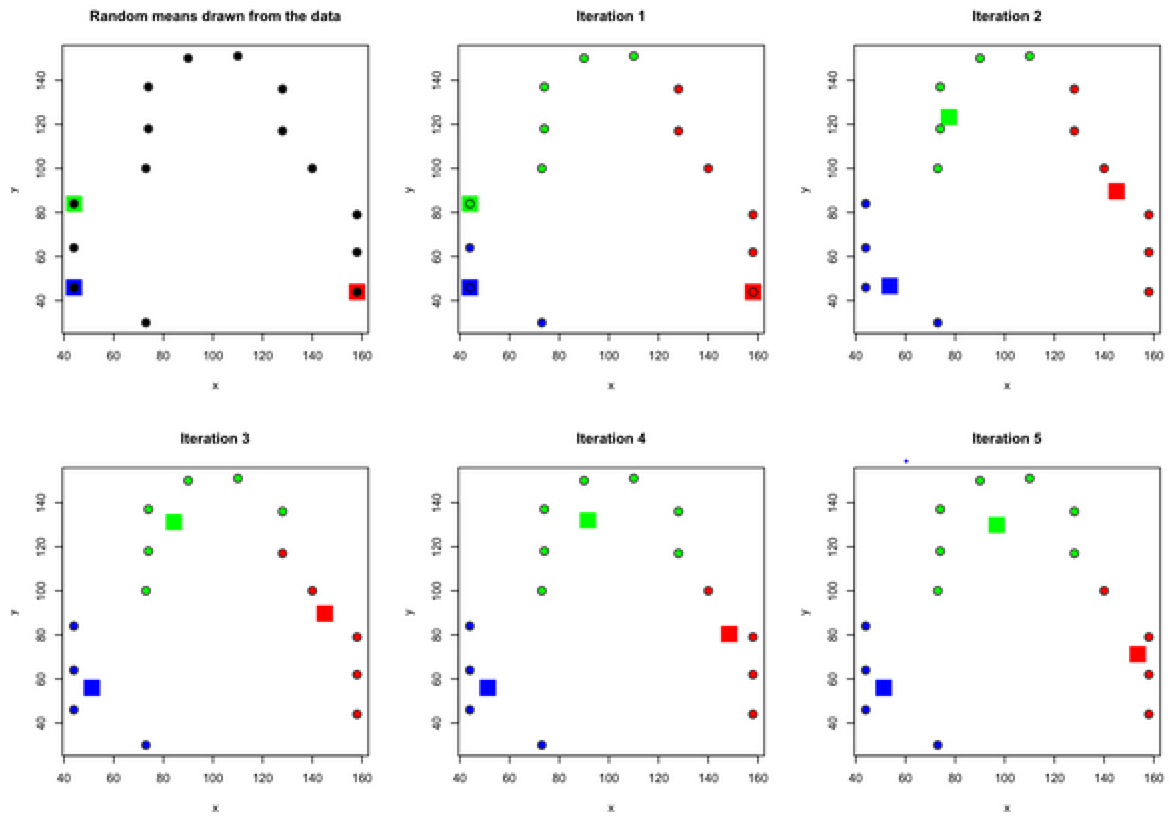

Figure 6. K-Means Clustering [20]

2.2. Advantages & Disadvantages

The main advantages of this algorithm are its simplicity and speed, which allows it to run on large datasets. Its disadvantage is that it does not yield the same result with each run, since the resulting clusters depend on the initial random assignments. It minimizes intra-cluster variance, but does not ensure that the result has a global minimum of variance. Another disadvantage is the requirement for the concept of a mean to be definable, which is not always the case. [21]

2.2.1. Initial Random Assignment Problem and K-Means++

K-means algorithm begins with k arbitrary "centers", typically chosen uniformly at random from the data points. Each point is then assigned to the nearest center, and each center is recomputed as the center of mass of all points assigned to it. These last two steps are repeated until the process stabilizes. One can check that change in cluster formation is monotonically decreasing, which ensures that no configuration is repeated during the course of the algorithm. Since there are only k^n possible clustering results, the process will always terminate. It is the speed and simplicity of the k-means method that makes it appealing but not its accuracy.

Indeed, there are many natural examples for which the algorithm generates arbitrarily bad clustering results (i.e., Iteration is unbounded even when n and k are fixed). This does not rely on an adversarial placement of the starting centers, so ultimate result may differ depending on initial centroid placement even time required to form same number of cluster with same data points be different.

David Arthur and Sergei Vassilvitski [11] proposed a variant that handles this initial randomized centroid choice issue. The new algorithm chooses initial centers depending on a particular mathematical function and assigns the datapoints depending on their squared distance to the centers already chosen. Choosing centers in this way is both fast and simple, and it already achieves guarantees that the final result would be same, whereas k-means cannot guarantee this. The authors proposed this technique to seed the initial centers for k-means, leading to a combined algorithm and named it k-means++. [22]

3. PARALLEL K-MEANS++ (WITH EQUIDISTANT CENTERS) ALGORITHM

The k-means algorithm has maintained its popularity due to its simple iterative nature. Given a set of cluster centers, each point can independently decide which center is closest to it, and given an assignment of points to clusters, computing the optimum center can be done by simply averaging the points. Indeed parallel implementations of k-means are readily available (see, for example MAHOUT at cwiki.apache.org/MAHOUT/k-means-clustering.html).

However, from a theoretical standpoint, k-means turns out to be not always a good clustering algorithm in terms of efficiency or quality: the running time can be exponential in the worst case [23] and even though the final solution is locally optimal, it can be very far away from the global optimum (even under repeated random initializations). Nevertheless, in practice the simplicity of k-means cannot be beat. Therefore, recent work has focused on improving the initialization procedure: deciding on a better way to initialize the clustering dramatically changes the performance of the K-Mean's iteration, both in terms of quality and convergence properties. An important step in this direction was taken by Ostrovsky et al. [24], and Arthur and Vassilvitskii [22], who showed a simple procedure that both leads to good theoretical guarantees for the quality of the solution, and, by virtue of a good starting point, improves upon the running time of K-Mean's iteration in practice. K-means++ [22], the algorithm selects only the first center uniformly at random from the data. Each subsequent center is selected with a probability proportional to its contribution to the overall error given the previous selections. Intuitively, the initialization algorithm exploits the fact that a good clustering is relatively spread out, thus when selecting a new cluster center, preference should be given to those further away from the previously selected centers. Formally, one can show that the k-means++ initialization leads to an O(log k) approximation of the optimum [22], or a constant approximation if the data is known to be well "clusterable" [24]. The experimental evaluation of the k-means++ initialization and the variants that followed [25] [26] [27] demonstrated that correctly initializing K-Mean's iteration is crucial if one were to obtain a good solution not only in theory, but also in practice. On a variety of datasets the k-means++ initialization obtained order of magnitude improvements over the random initialization.

However, the downside of the k-means++ initialization is its inherently sequential nature. Although its total running time is O(nkd), when looking for a k-clustering of n points in $R^{d,}$ is the same as that of a single K-mean's iteration, it is not apparently parallelizable. The probability with which a point is chosen to be the i^th center depends critically on the realization of the previous i−1 centers (it is the previous choices that determine which points are away within the current solution). A naive implementation of the k-means++ initialization makes k passes over the data in order to produce the initial centers. This fact is exacerbated in the massive data scenario. First, as datasets grow, so does the number of classes into which one wishes to partition the data. For example, clustering millions of points into k = 100 or k = 1000 is typical, but a k-means++ initialization would be very slow in these cases. This slowdown is even more detrimental when the rest of the algorithm can be implemented in a parallel environment like MapReduce [7]. For many applications it is desirable to have an initialization algorithm with similar guarantees to k-means++ that can be efficiently parallelized. [28] [29]

3.1. Our Contribution

In this work, we have implemented a parallel version of the k-means++ (with Initial Equidistant Centers) initialization algorithm and empirically demonstrate its practical effectiveness. The main idea is that, in each iteration, it would require a total of (nk) distance computations where n is the number of objects and k is the number of clusters being created. It is obvious that the distance computations between one object with the centers is irrelevant to the distance computations between other objects with the corresponding centers. Therefore, distance computations between different objects with centers can be executed in parallel. In each iteration, the new centers, which are used during the next iteration, should be updated. Hence, the iterative procedure must be executed sequentially. This initialization algorithm, which we call k-means++ (with Initial Equidistant Centers), is quite simple and lends itself to easy parallel implementation. We then evaluate the performance of this algorithm using datasets.

Our key observations in the experiments are:

- The parallel implementation of k-means++ (with Initial Equidistant Centers) is much faster than existing parallel algorithms for k-means because of less number of iterations. As initial centroids are fixed and certain, the converging criteria are the same as for parallel k-means. In conventional k-means the parallel algorithm may iterate repeatedly because of drastic change of data points assignment to each cluster in each iteration (as the cluster centroid changes more than k-means++).
- The number of iterations until the K-Means algorithm converges is smallest when using k-means++ (with Initial Equidistant Centers) as the seed.

3.2. The Algorithm

We now describe the parallel k-means++ (with Initial Equidistant Centers) algorithm using the Map-Reduce approach. It starts with the initialization step.

3.2.1. Initialization Function

As part of the initialization, we have used a mathematical function, which selects the initial centroids at equal distance depending on the number of clusters entered as input. The pseudo code is represented in Algorithm 1.

Algorithm 1. Initialization

```
Input: Text file contains the data points; Number of Clusters
Output: Centroid co-ordinates for each cluster

For i=0 to cluster.length
    Centroid_X-Axis = (((MaxXValue - MinXValue) / (clusters.length + 1)) * i) +
    MinXValue;
    Centroid_Y-Axis = (((MaxYValue - MinYValue) / (clusters.length + 1)) * i) +
    MinYValue;
End For
```

3.2.2. Map Function

As discussed earlier, the map class takes the input dataset stored in HDFS and the number of clusters to be made as input and assigns each data point to the most relevant cluster.

Algorithm 2. Map (*key, value)*

```
Input: text file containing data set
Output: <key', value'> pair, where the key' is the index of the closest center point and value'
is a datapoint
   1.  If iteration = First
          {
            Choose the centroid at equal distance by reading the entire input dataset
          }
       Else
         {
            Read the centroid file created by Reduce method and assign each individual
            centroid to individual cluster
       }
  2.  tempEuDT = Double.MAX VALUE;
  3. index = -1;
  4. For i=0 to centers.length do
       dis= ComputeEucledianDist(instance, centers[i]);
       If dis < tempEuDT
          {
            tempEuDT = dis;
            index = i;
          }
  5. End For
  6. Take Centroid as key';
  7. Construct value' as individual datapoint;
  8. Output < key, value> pair;
  9. End
```

Note that Step 2 and Step 3 initialize the auxiliary variable *tempEuDT* and *index.*

3.2.3. Reduce Function

The Reduce function combines the datapoints corresponding into a single centroid to a single cluster. Then by calculation the mean of each cluster, it decides the new centroid clusters to be used as input and assigns each data point to the most relevant cluster.

Algorithm 3. *Reduce* (*key*, *value)*

```
Input: key is the centroid of the cluster, Value is the list of the datapoints from different nodes
Output: < key, value> pair, where the key' is the index of the cluster, value' datapoint.
1.  Assign each datapoint corresponding to a single centroid to an arraylist;
2.  For i=0 to length of arraylist
            Sum all the value;
    End For
3. Take the mean of datapoints;
4. Add the mean value to the centroid Text file;
5. If new centroid== Old centroid
            Set as converged
   Else
            Go for next iteration
   End If
6. Output < key, value> pair;
7. End
```

The algorithm iterates until the converging criteria meets. In this paper, the algorithm converges when the location of the current centroid is same as the location of the previous centroid.

In every iterations the reduce method creates the centroid.txt file and saves it to the HDFS, and the file is used as the input file for the Map method for the second iteration.

4. EXPERIMENTAL SETUP AND RESULTS

In this section, we present the experimental setup for evaluating MapReduce-enabled k-means++ (with Initial Equidistant Centers). The parallel algorithm was run using the departmental Hadoop cluster.

4.1. Datasets

To validate the quality of the algorithm and its implementation, we have mainly selected the parameters:

- Performance of the algorithm on a huge data set
- Purity of clusters obtained as output
- Scalability of the implementation with respect to increasing number of nodes.

The real datasets that are used are the following:

Iris: We took this dataset from the UCI machine learning repository [30]. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Balance: We took this dataset from the UCI machine learning repository [30]. This data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight, and the right distance. The correct way to find the class is the greater of (left-distance * left-weight) and (right-distance * right-weight). If they are equal, it is balanced. Balance dataset contains total 625 instances with 3 class labels.

Ecoli: We took this dataset from the UCI machine learning repository [30]. This dataset is about protein and bacteria. [31] [32]This dataset contains 336 instances with 8 class labels.

Glass: We took this dataset from UCI machine learning repository [30].This dataset contains 214 instances with 7 class labels. Vina Spiehler, [33] conducted a comparison test of her rule-based system, BEAGLE, the nearest-neighbor algorithm, and discriminant analysis. BEAGLE is a product available through VRS Consulting, Inc.

Mouse: We took this dataset from the UCI machine learning repository [30]. The data set consists of the expression levels of 77 proteins/protein modifications that produced detectable signals in the nuclear fraction of cortex. There are 38 control mice and 34 trisomic mice (Down syndrome), for a total of 72 mice. In the experiments, 15 measurements were registered of each protein per sample/mouse. Therefore, for control mice, there are 38x15, or 570 measurements, and for trisomic mice, there are 34x15, or 510 measurements. The dataset contains a total of 1080 measurements per protein and 8 class labels. The eight classes of mice are described based on features such as genotype, behavior and treatment. According to genotype, mice can be control or trisomic. According to behavior, some mice have been stimulated to learn (context-shock) and others have not (shock-context) and in order to assess the effect of the drug memantine in recovering the ability to learn in trisomic mice, some mice have been injected with the drug and others have not.

Seeds: We took this dataset from the UCI machine learning repository [30]. The examined group comprised kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian. High quality visualization of the internal kernel structure was detected using a soft X-ray technique. It is non-destructive and considerably cheaper than other more sophisticated imaging techniques like scanning microscopy or laser technology. The images were recorded on 13x18 cm X-ray KODAK plates. Studies were conducted using combine harvested wheat grain originating from experimental fields, explored at the Institute of Agrophysics of the Polish Academy of Sciences in Lublin.

Dense: This dataset contains 150 instances with 3 class labels. This dataset is also taken from UCI machine learning repository [30].

Table 1 describes the datasets:

Table 1. Real Dataset

Dataset	Instances	Class Labels	Area
Iris	150	3	Life
Balance	625	3	Social
Ecoli	336	8	Life
Glass	214	7	Physical
Seed	210	3	Life
Mice	1080	4	Life
Dense	150	3	Social

Synthetic: We have used the datasets generated in [34]. These synthetic datasets have been generated using the data generator [35]. The dataset ranges from 0.5 million to 4 million. In order to characterize the synthetic datasets, each name has a specific pattern: Number of data, number of dimensions and number of clusters shown in Table 2.

Table 2. Synthetic Dataset

Dataset	#Records	#Dimension	Size(MB)	Type	#Clusters
Dataset1	0.5	2	19.07	Synth	5
F1m2d5c	1	2	41	Synth	5
F2m2d5c	2	2	83.01	Synth	5
F4m2d5c	4	2	163.05	Synth	5

4.2. Evaluation Measure

In this paper, we use the purity measure for the evaluation of the cluster quality [36], which is the standard measure of clustering quality and is calculated as:

$$Purity = \frac{1}{n}\sum_{j=1}^{k} \max(|Li \cap Cj| : i = 1,2,3 \ldots . q) \tag{3}$$

Where L_i denotes the true assignments of the data instances in cluster i; q is the number of actual clusters in the data set. A clustering algorithm with large purity values indicates better clustering solutions. The clustering quality is perfect if the clusters only contain data instances from one true cluster; in that case the purity is equal to 1.

We have used the speedup measure to measure the performance of the algorithm by continuously increasing the number of nodes and keeping the dataset size the same. [37]

$$Speedup = \frac{T_2}{T_n} \tag{4}$$

Where T_2 is the running time using 2 nodes, and Tn is the running time using n nodes, where n is a multiple of 2.

Lastly we have used scaleup, which is a measure of speedup that increases with increasing dataset sizes to evaluate the ability of the parallel algorithm utilizing the cluster nodes effectively.

4.3. Test Cases

4.3.1. Test Case 1

We ran the algorithm for four different datasets from 0.5 million records to 4 million records to build 5 clusters each time. All the datasets are of 2 dimensional data.

4.3.2. Test Case 2

We ran the algorithm for 7 different datasets contain various types of data to measure the purity of cluster. Then we compare the output with Table 3.

Table 3. Cluster Purity Table

Dataset	**K-Means**	**HC**	**FF**	**LVQ**
Iris	0.887	0.887	0.860	0.507
Ecoli	0.774	0.654	0.599	0.654
Glass	0.542	0.463	0.481	0.411
Balance	0.659	0.632	0.653	0.619
Seeds	0.876	0.895	0.667	0.667
Mouse	0.827	0.91	0.800	0.843
Vary Density	0.953	0.667	0.667	0.567

4.3.3. Test Case 3

We ran our algorithm on the dataset containing 2 million records, however we continuously increased the number of nodes e.g. 2, 4, 8,…, 16. Then by comparing the running time we could able to measure the scalability of the algorithm.

4.4. Results

4.4.1. Test Case 1

When we ran the test case 1, we found with the growth of the dataset size (from 0.5 million to 4 million) the time taken by the algorithm is increasing in a slower rate. The statistics is given in Table 4.

Table 4. Performance Testing Results

Number of Records (million)	Time taken (in milliseconds)
0.5	197166
1	201563
2	233865
4	300989

If we put the result in graphical format, we can find the graph (Figure 7) supports our statement.

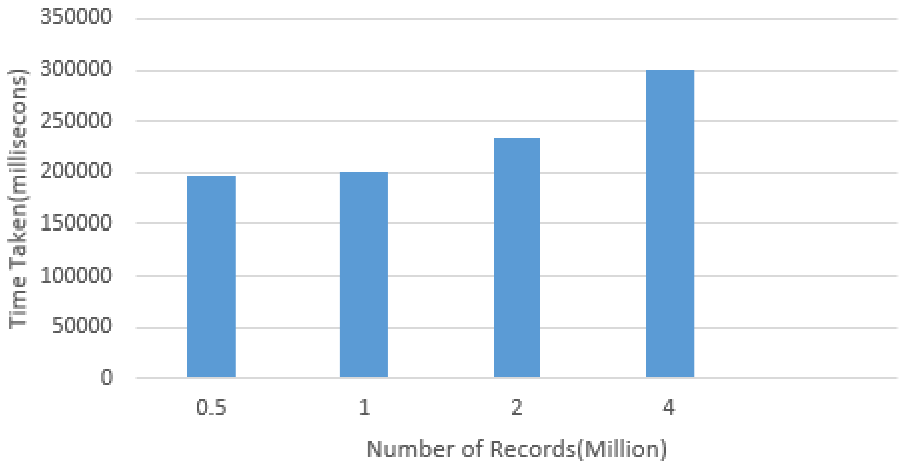

Figure 7. Performance Graph

4.4.2. Test Case 2

We used seven two-dimensional data sets (as described in Table 8) to perform the purity check. In all those data sets the true assignments (each data point is labeled to one of the centroids initially) of the clusters are given by the last column of the data set. The assignment the clustering algorithm returns is the resulting assignment. And with that information, we have calculated the purity by comparing the resulting assignment with the given assignment (given by the data set).

Iris Dataset: Iris dataset has three different types: Iris-setosa, Iris-versicolor and Iris-virginica. Resulting assignment of Iris-setosa, Iris-versicolor and Iris-virginica are 50, 45 and 40, respectively and the total number of data points is 150. So, the purity is 0.9 (by the below calculation).

(50+45+40)/150 = 0.9.

Balance Dataset: Balance dataset has three different types: L, R and B. Resulting assignment of L, R and B are 116, 113 and 154, respectively. The total number of data points is 625. Thus, the purity is 0.695.

(166+113+154)/625 = 0.692.

Ecoli Dataset: Ecoli dataset has eight different types: cp (cytoplasm), im (inner membrane without signal sequence), pp (perisplasm), imU (inner membrane, uncleavable signal sequence), om (outer membrane), omL (outer membrane lipoprotein), imL (inner membrane lipoprotein), imS (inner membrane, cleavable signal sequence) and the resulting assignments are 5, 46, 72, 29, 27, 29, 35, 23, respectively. That gives a purity as 0.791 for 336 members.

(5+46+72+29+27+29+35+23)/336 = 0.791.

Mice Dataset: Mice dataset has four different types: Ear-Left, Head, Ear-Right and noise. The resulting assignments are 100, 144, 88 and 100, respectively. Total number of data is 500. The purity of the dataset is 0.864.

(100+144+88+100)/500= 0.864.

Dense Dataset: Vary dense dataset contains Cluster1, Cluster2 and Cluster3 and their resulting assignment counts are 93, 24 and 26 out of 150 in total.

(93+24+26)/150 = 0.953.

Glass Dataset: Glass has seven different types from 1 through 7 and their resulting assignment according to the algorithm is 1,50,19,28,13,1 and 4, respectively. So when we put these in the purity equation the result is 0.586 for 214 total glass data.

(1+50+19+28+13+1+14)/214 = 0.586.

Seeds Dataset: Seeds contains 1, 2 and 3 as type and their resulting assignment is 78, 73 and 26. The purity of this data set is 0.850.

(78+73+26)/208=0.850.

We have compared the calculations with the cluster purity table. Table 5 describes the comparison.

Table 5. Cluster Purity Table for K-Means++ (With Initial Equidistant Centers)

Dataset	**Kmeans**	**Kmeans++**
Iris	0.887	**0.900**
Ecoli	0.774	**0.791**
Glass	0.542	**0.568**
Seed	**0.876**	0.850
Balance	0.659	**0.692**
Mouse	0.827	**0.864**
Vary Density	0.953	**0.953**

From the result we can see that mostly the purity is >8.0, which is very good in quality, and our algorithm has outperformed k-means in most of the cases. Figures 10-16 show the comparisons among K-Means, HC, FF, LCQ and K-Means++ algorithm with respect to their purity calculations (refer to Table 3 and Table 5).

Figures 8 through 14 represent the purity comparison among K-means algorithm, HC algorithm, FF algorithm, LVQ algorithm and K-Means++ algorithm.

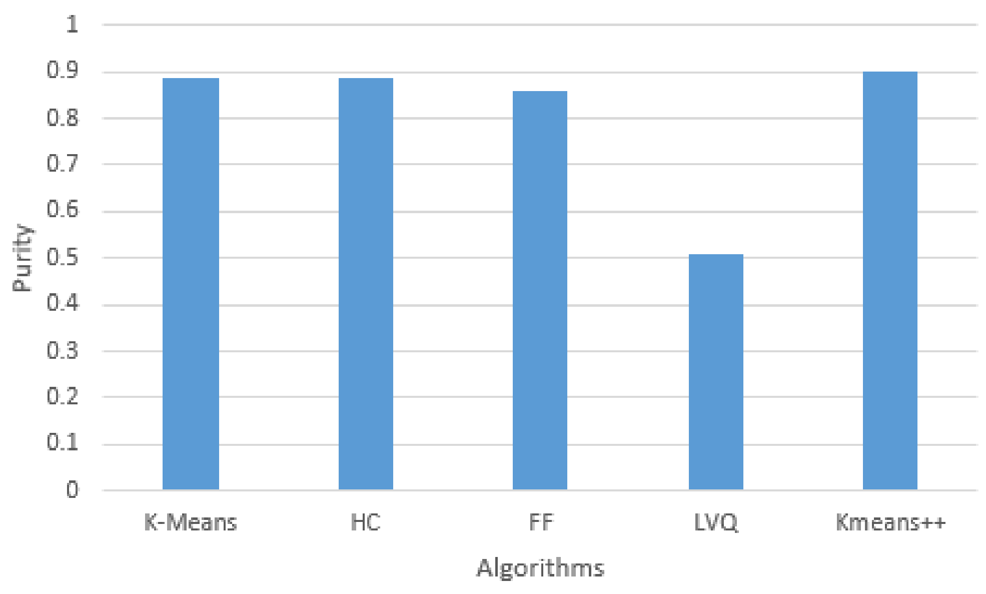

Figure 8. Iris Data Set Purity Comparison

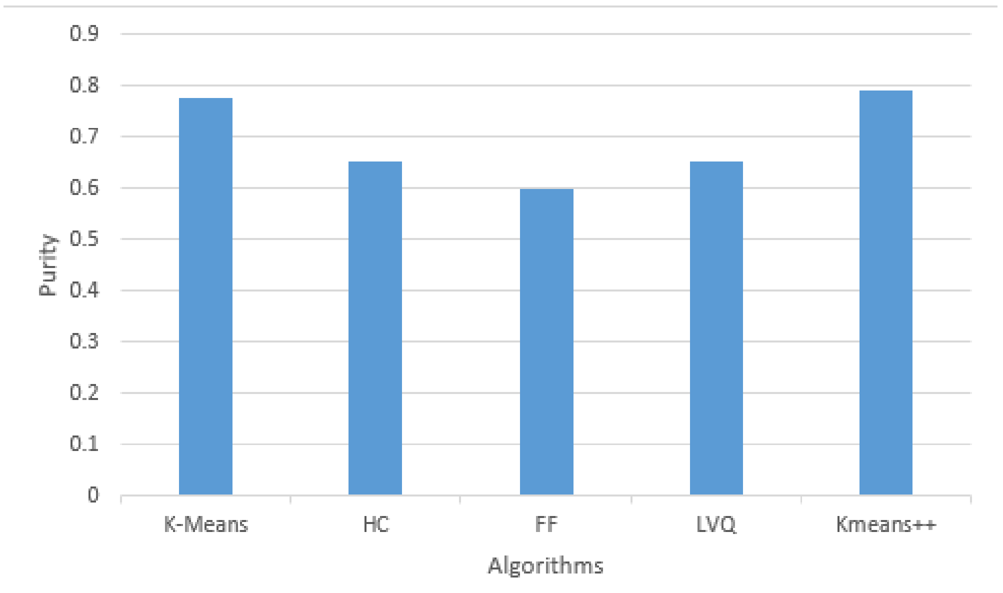

Figure 9. Ecoli Dataset Purity Comparison

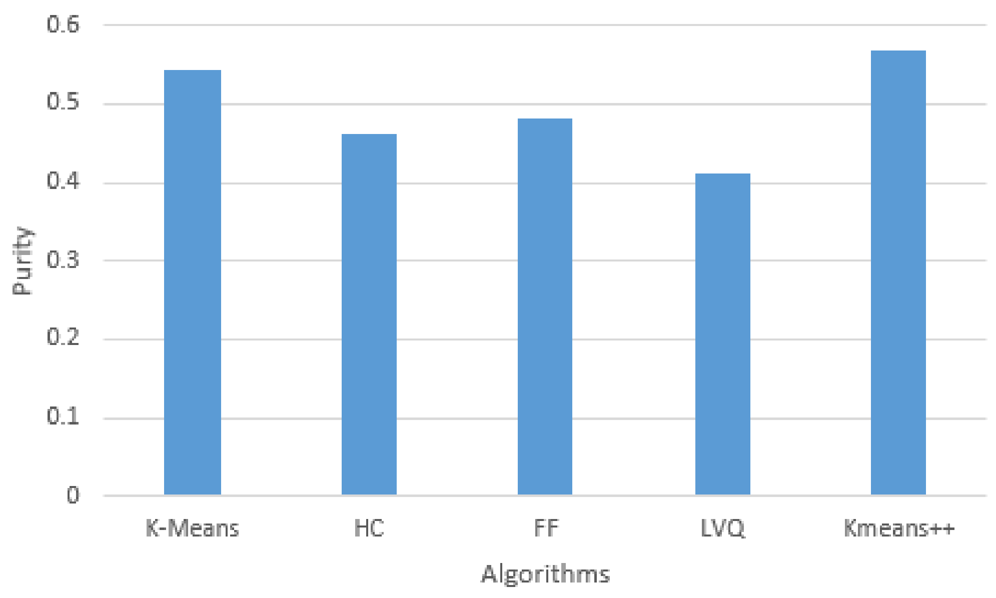

Figure 10. Glass Dataset Purity Comparison

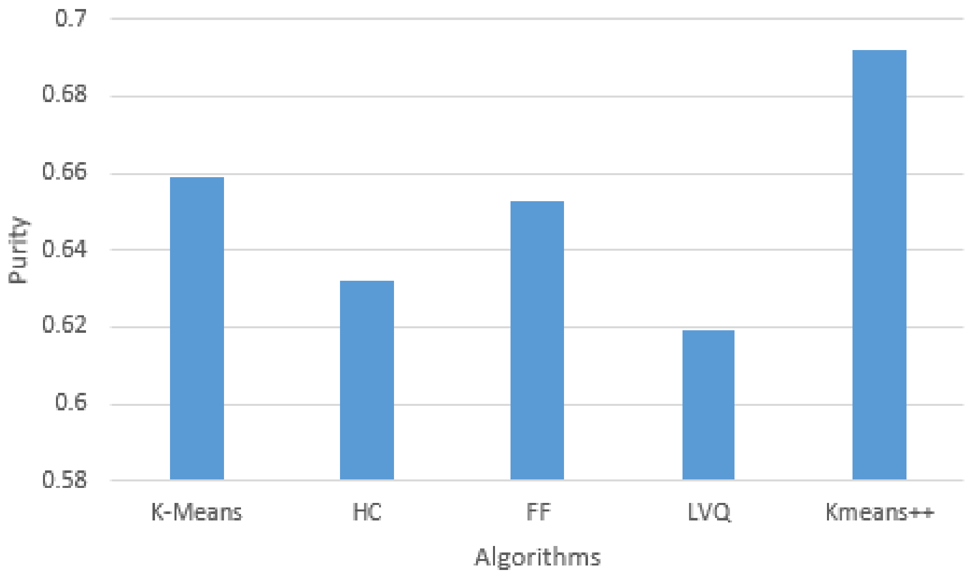

Figure 11. Balance Dataset Purity Comparison

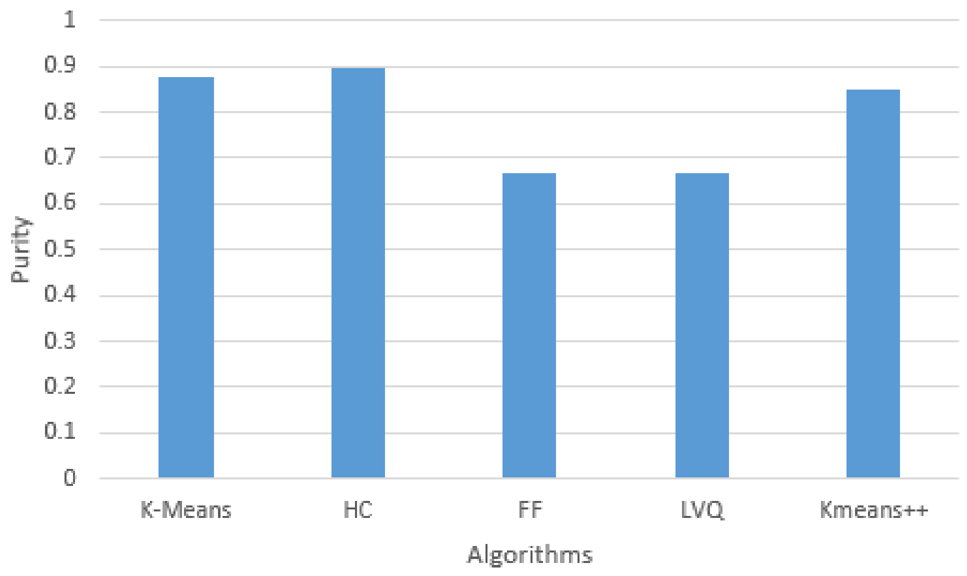

Figure 12. Seed Dataset Purity Comparison

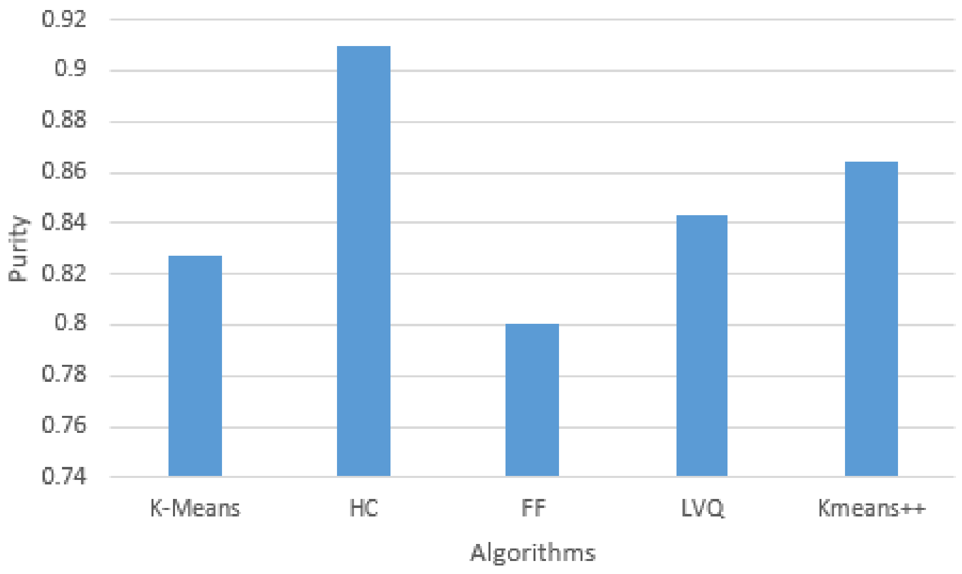

Figure 13. Mouse Dataset Purity Comparison

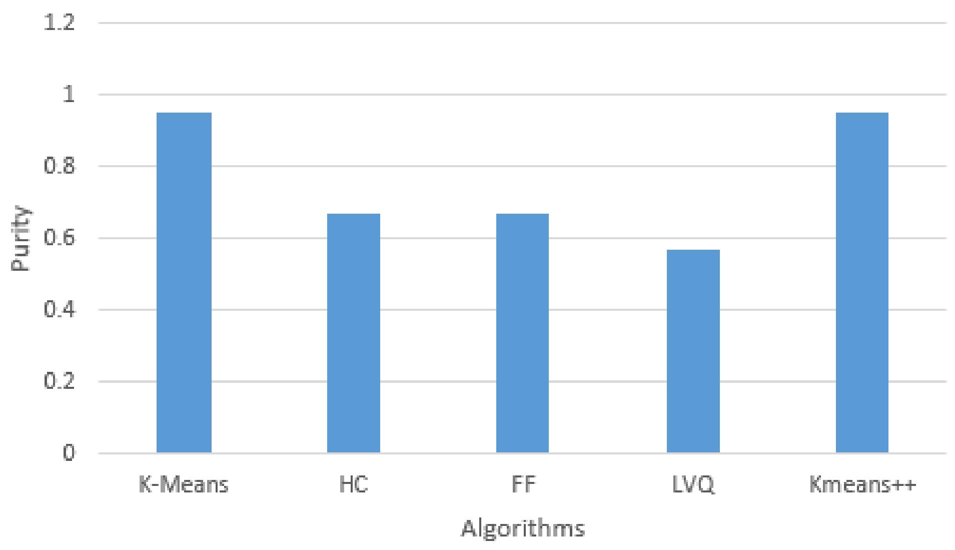

Figure 14. Vary Density Dataset Purity Comparison

4.4.3. Test Case 3

We used the F2m2d5c dataset to check the scalability of the algorithm with respect to increasing the number of nodes from 2 to 16. We found that time required to run the algorithm decreases (except for one exception) with increasing number of nodes and keeping all other inputs the same each time. That confirms the scalable nature of the algorithm. Table 6 is the result, we gathered from the scalability testing.

Table 6. Scalability Test Result

Number of Nodes	Time (in milliseconds)
2	310084
4	272902
6	243619
8	243325
10	233584
12	235169
14	201563
16	197166

Figure 15 shows how the time required decreases with increasing number of nodes.

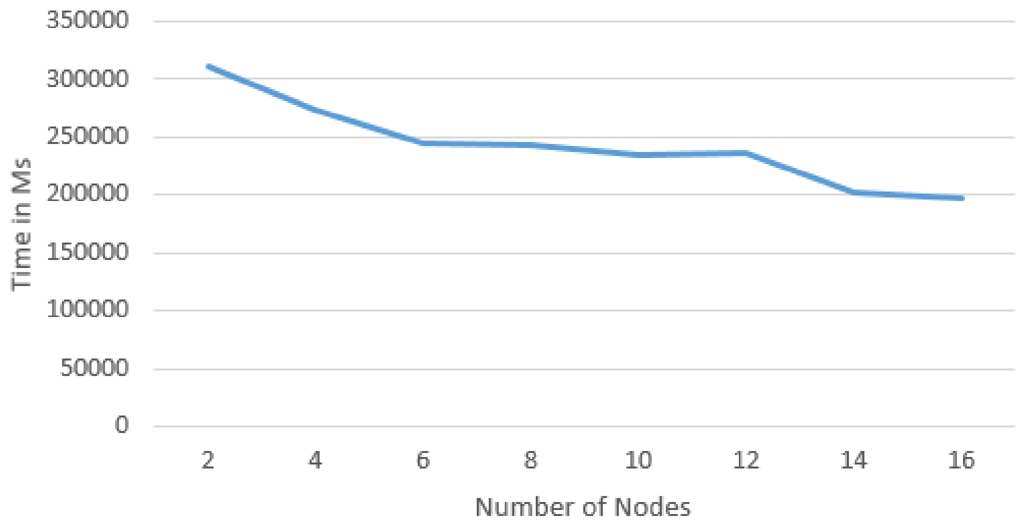

Figure 15. Scalability

5. CONCLUSIONS AND FUTURE WORK

As data clustering has attracted a significant amount of research attention, many clustering algorithms have been proposed in the past decades. However, the enlarging data in applications makes clustering of very large scale of data a challenging task.

In this paper, we propose a fast parallel k-means++ (with Initial Equidistant Centers) clustering algorithm based on MapReduce. This algorithm is an improvement over the conventional K-Means clustering algorithm. In K-Means, the initial centroid selection procedure is purely random and that hugely affects the overall performance of the algorithm. K-Means++ improves the uncertainty of the performance by selecting initial centroids using a predefined rule. In this paper, to set the initial centroids we have divided the whole dataset into a certain number of clusters first and then from those chunks of data we have taken, e.g. in the 2-dimensional case, the average of the X-coordinate (by subtracting the minimum X from the maximum X and then divide that by length of that chunk) and the average of the Y-coordinate (using the same rule as the X-coordinate) for each and every chunk. Then, once the initial centroids are set, the rest of the process iterates over the entire data set and assign data to each cluster depending on the minimum Euclidean distance from datapoints to cluster centroid. Then, once all the datapoints are assigned to the clusters, the new cluster center gets recalculated and again the assignment process takes place. Iteration occurs till the current coordinates of all centroids are same as previous cycle (the converging criteria). These iterations are taking place in parallel following the MapReduce concept because clusters are independent of each other, so datapoints assignment or centroid recalculation can take place in parallel for all clusters. Thus, the K-means++ algorithm was parallelized using the MapReduce paradigm.

From the result of our experiments, we found the purity of the clusters produced by this algorithm are very good. Moreover, the performance is good and the result is constant for every time we run the algorithm as the initial centroids are always the same for a particular dataset. Also, the parallel implementation makes the algorithm robust and scalable.

Our future work aims to find a mechanism to work with various types of data files, so that we can use any data files types for the testing without formatting it in a particular text format. Also, we are aiming to apply it on some real data set associated with a real life problem.

6. REFERENCES

[1] A. K. Jain, M. N. Murthy and P. J. Flynn, "Data Clustering: A Review," *ACM Computing Surveys (CSUR),* vol. 31, no. 3, pp. 264-323, Sept. 1999.

[2] "Cluster Analysis," [Online]. Available: https://en.wikipedia.org/wiki/Cluster_analysis.

[3] A. Katal, M. Wazid and R. Goudar, "Big data: Issues, challenges, tools and Good practices," in *Contemporary Computing (IC3), 2013 Sixth International Conference,* Noida, 8-10 Aug. 2013.

[4] M. Wattenberg, V. B. Fernanda and K. Hollenbach, "Visualizing Activity on Wikipedia with Chromograms," *Human-Computer Interaction – INTERACT*, vol. 4663, pp. 272-287, 2007.

[5] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh and A. Byers, "Big data: The next frontier for competition," *McKinsey Global Institute*, 2011.

[6] A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A. Zomaya, S. Foufou and A. Bouras, "A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis," *Emerging Topics in Computing, IEEE Transactions*, vol. 2, no. 3, pp. 267 - 279, 2014.

[7] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM - 50th anniversary issue: 1958 - 2008,* vol. 51, no. 1, pp. 107-113, 2008.

[8] The Apache Software Foundation, [Online]. Available: https://hadoop.apache.org/.

[9] "Hadoop," [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html.

[10] T. Gunarathne and S. Perera, Recipes for analyzing large and complex datasets with Hadoop MapReduce, *Livery Place: Packt Publishing Ltd*, 2013.

[11] Z. Huang, "Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values," *Data Mining and Knowledge Discovery* , vol. 2, no. 3, pp. 283-304, 1988.

[12] J. B. MacQueen, "Some Methods for classification and Analysis of Multivariate Observations," in *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability,* Berkeley, University of California Press, 1967, pp. 281-297.

[13] H. Steinhaus, "Sur la division des corps matériels en parties," *Bull. Acad. Polon. Sci. (in French),* vol. 12, no. 4, pp. 801-804, 1957.

[14] S. p. Llyod, "Least square quantization in PCM," *Bell Telephone Laboratories Paper,* vol. 28, no. 2, pp. 129-137, 1982.

[15] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory,* vol. 2, no. 28, pp. 129-137, 1982.

[16] E. Forgy, "Cluster analysis of multivariate data: efficiency versus interpretability of classifications," *Biometrics,* vol. 21, pp. 768-768, 1965.

[17] J. Hartigan, Clustering algorithms, John Wiley & Sons, Inc, 1975.

[18] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-Means Clustering Algorithm," *Journal of the Royal Statistical Society, Series C,* vol. 28, no. 1, pp. 100-108, 1979.

[19] A. Naik, "K-Means Clustering," [Online]. Available: https://sites.google.com/site/dataclusteringalgorithms/k-means-clustering-algorithm.

[20] E. Forster, G. Wallas and A. Gide, "Data Visualization," [Online]. Available: https://apandre.wordpress.com/visible-data/cluster-analysis/.

[21] S. Kumari and A. Kaushik, "A Survey of Clustering with Optimized k-Medoids Algorithm," *International Journal of Computational Engineering & Management,*, vol. 2, no. 4, pp. 121-128, 2014.

[22] D. Arthur and V. Sergei, "k-means++: The Advantages of Careful Seeding," *SODA '07 Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms,* pp. 1027-1035, 2007.

[23] D. Arthur and V. Sergei, "How slow is the k-means method?," *SOCG,* p. pages 144–153, 2006.

[24] R. Ostrovsky, Y. Rabani, L. Schulman and C. Swamy, "The effectiveness of Lloyd-type methods for the k-means problem," *FOCS,* pp. 165-176, 2006.

[25] M. Ackermann, C. Lammersen, M. M¨artens, C. Raupach, C. Sohler and K. Swierkot, "StreamKM++: A clustering algorithm for data streams," *ALENEX,* p. 173–187, 2010.

[26] N. Ailon, R. Jaiswal and R. Monteleoni, "Streaming k-means approximation," *NIPS,* p. 10–18, 2009.

[27] A. Ene and B. Moseley, "Fast clustering using MapReduce," *KDD,* pp. 681-689, 2011.

[28] F. Farnstrom, J. Lewis and C. Elkan, "Scalability for clustering algorithms revisited," *SIGKDD Explor.,* vol. 2, pp. 51-57, 2000.

[29] B. Bahmani, B. Moseley, A. Vattani, R. Kumar and S. Vassilvitskii, "Scalable k-Means++," *Proceedings of the VLDB Endowment,* vol. 5, no. 7, 2012.

[30] "UC Irvine Machine Learning Repository," [Online]. Available: http://archive.ics.uci.edu/ml/index.html.

[31] K. Nakai and M. Kanehisa, "Expert Sytem for Predicting Protein Localization Sites in Gram-Negative Bacteria," *PROTEINS: Structure, Function, and Genetic,* vol. 11, pp. 95-110, 1991.

[32] K. Nakai and M. Kanehisa, "A Knowledge Base for Predicting Protein Localization Sites in Eukaryotic Cells," *Genomics,* no. 14, pp. 897-992, 1992.

[33] V. Spiehler, "Computer-assisted interpretation in forensic toxicology: morphine-involved deaths," *Journal of Forensic Sciences,* vol. 34, no. 5, pp. 1104-1115, 03 01 1989.

[34] N. Al-Madi, b. IAljarah and S. Ludwig, "Parallel Glowworm Swarm Optimization Clustering Using Mar Reduce," *Swarm Intelligence (SIS), 2014 IEEE Symposium on*, Orlando,FL, 2014.

[35] R. Orlandic, Y. Lai and W. Yee, "Clustering the high-dimensional data using an efficient and effective data space reduction," *Proc. ACM 14th Conf. on Information and knowledge Management,* pp. 201-208, 2005.

[36] Y. Zhao and G. Karypis, "Evaluation of hierarchical clustering algorithm for documents Dataset," *CIKM'02,* vol. 11, p. 515–524, 2002.

[37] N. Al-Madi, I. Aljarah and S. Ludwig, "Parallel glowworm swarm optimization clustering algorithm based on MapReduce," in *Swarm Intelligence (SIS), 2014 IEEE Symposium on*, Orlando, FL, 2014.