— Parallel K-means using MapReduce - IPKMeans vs. PKMeans —

Patrick Killeen
School of Computer Science
University of Ottawa
Ottawa, Canada K1N 6N5
pkill013@uottawa.ca

December 11, 2018

Abstract

With the increasing trend of the Internet of Things, Big Data is becoming an issue for modern data analytics. Big Data analytics is a popular research field that attempts to manage all this data. MapReduce is a popular framework for solving problems that are involved with huge datasets. The K-means clustering algorithm is an example of a data analytics algorithm. A lot of research has been put into implementing K-means in parallel using Hadoop's MapReduce. IPKMeans and PKMeans are two algorithms that attempt to run K-means in parallel using MapReduce. IPKMeans is an improvement of PKMeans. This work runs experiments on theses two algorithms on a Hadoop cluster and verifies the results and claims discussed in the paper that proposes IPKMeans.

1 Introduction

Computing and storage has become cheaper over the years. There is also more and more data being created each day [11]. There is so much data that data analytics and knowledge discovery is becoming more feasible [3]. However, standard data mining algorithms cannot cope with this increasing amount of data [29]. This is pushing the Big Data analytics research field [32].

Parallel computing can be used to speed up algorithms. To speed up an algorithm using parallelism, it must be converted from serial to parallel. There are frameworks designed to handle fault tolerance, distributed computing, synchronization, data storage and processing. Hadoop MapReduce is an example of such a framework [17]. However, it isn't always straight forward how to convert a serial algorithm to parallel. For example, MapReduce is a framework for processing Big Data in jobs, but it isn't designed for many iterations [5]. K-means clustering algorithm [5] is based on iterations [22].

This paper paper compares two approaches that implement parallel K-means using MapReduce, namely, PKMeans[38], and IPKMeans[18]. This paper is organized into the following sections: Literature Review, Problem Statement, Hadoop Cluster Configuration, Results, and Conclusion.

2 Literature Review

2.1 Big Data

The growth of data creation is increasing at an exponential rate [11] [16]. Big Data is becoming a problem for processing large amounts of data with limited processing power [9][14]. Data mining algorithms need to be re-designed to handle such massive amounts of data. Big Data analytics is the field of performing machine learning on Big Data. According to [4], in Big Data there are three Vs, namely: Volume, Velocity, and Variety. Velocity represents the speed at which the data is flowing(a fast data stream). Volume represents the sheer size of the data. Variety represents the heterogeneous nature of the data, as data is coming from many data sources. There are two more dimensions: Variability and Complexity. Variability represents the difference of flow between data source. Complexity must also be taken into account [4].

With all this data it allows us to acquire knowledge [3]. The trend of huge amounts of data creation has encouraged the research field of Big Data analytics [6][10][32]. Big Data leads to efficient storage and processing requirements [20] in a distributed manner [9]. Therefore, efficient processing algorithms are required for analysing data [20]. Big Data analytics can tackled by using parallelization [10]. For example by using the Hadoop framework [16][31]. Clustering is an important aspect of Big Data analytics [31].

2.2 Data Mining and Clustering

Data mining consists of extracting knowledge and finding novelties from datasets by analysing patterns among the data elements [8][31]. Clustering is a data mining technique [4] [3][29], which is an unsupervised approach for finding outliers and grouping data together [9]. Clustering a dataset partitions it into groups of similar data elements [8][11][12], such that each group/cluster have data elements that are different from data elements in other clusters [3][13] [16]. Clustering is a popular technique for trying to solve data analytics problems [10][14]. It has applications such as information retrieval [16], stock exchange analysis [21], opinion mining [28], and image pattern recognition [34]. Knowledge discovery using data analytics can be divided into stages: data preprocessing, clustering the preprocessed data, and analysing the results for interesting patterns [26]. It becomes more difficult to perform data analytics as datasets become larger [11][13] [21][22][29] [35].

2.3 K-means

K-means algorithm is an efficient clustering algorithm [5] [7][12][33] [35][36]. K-means is one of the most popular unsupervised clustering algorithms [6][5] [10][11][15] [18][23], because of its simplicity and efficiency [12][14][31]. It can be used for clustering large datasets [15], which can be made of structured or unstructured data [8]. It has many different applications [7][23].

K-means attempts to cluster datasets into k clusters of similar elements [3]. It first starts the initialization stage by choosing k centroids (the centers of clusters) at random [22][25], and assigns each data element of the dataset to the nearest centroid using the Euclidean distance. At the end of this stage a new set of k centroids are calculated by taking average Euclidean distance of each data element within a cluster. This is done until the centroids converge [18][4]. Most of the computations performed in this algorithm are

distance computations, which are performed when comparing all data elements' distance to each of the centroids [5].

However, the k-means isn't perfect, it has its limitations. It has trouble dealing with outliers [11][25]. That is, the algorithm assigns each data element to a cluster, but it may not make sense in some contexts to assign an outlier to a cluster. Furthermore, the resulting clusters's stability and accuracy are sensitive to the initial centroids chosen [36]. That is, the resulting clusters will vary depending on the initial centroids chosen [22], and random initial centroids prove to yield unstable cluster results [3][25]. Therefore, to optimize k-means, care must be taken when choosing the initial centroids [12]. One of the bottle-necks in k-means is the number of iterations [22]. The more iterations there are, the more the clusters will converge at the cost of increased computations. As datasets becomes extremely large, k-means begins to lack in performance [6] [15][33][35], and its results become unstable [11]. The larger the dataset, the more iterations that will be required for high quality clusters, which will therefore take more computations [35]. Execution time could be improved using parallelism [12].

2.4 Map Reduce

As we enter the Big Data era, a lot of research is put into MapReduce [14]. MapReduce is a framework for processing Big Data [23] in parallel [7] [10][11] in a distributed manner. Map Reduce is a framework proposed by Google for processing Big Data, which involves storing, appending, and also running jobs seamlessly in a parallel, distributed, and fault tolerant manner [17]. Apache's Hadoop Map Reduce is written in Java and is open source [3], which is a version of Google's Map Reduce[15]. MapReduce has 2 phases, the Map phase and the Reduce phase [3][16][37]. User defined map and reduce functions are used to process key-value pairs as inputs [15]. Map Reduce partitions data into subsets during the mapping phase, and assigns each partition to a worker machine to be processed [35]. Once each worker has processed each partition, the results are combined in the reduction phase. The framework was created to meet the requirements of Big Data, trying to make sense of all these data available. It is a popular and is used by many companies. It can be deployed to many 100s of machines for processing Big Data [3]. There is no data cached between two consecutive MapReduce jobs [5]. There aren't very many data mining algorithms that are implemented using MapReduce [10]. The MapReduce jobs have a lot of I/O cost from reading and upon job completion writing to the file system. Therefore, many iterations in algorithms using MapReduce should be avoided when minimizing performance costs [33].

2.5 Hadoop Distributed File System

Hadoop framework also provides a distributed file system [10][15]. HDFS is in charge of storing and processing large amounts of data on distributed nodes [16], creating replications when necessary [3].

2.6 Parallel k-means using Map Reduce

For applying k-means to Big Data, k-means can be parallelized using MapReduce. There is a difficulty when combining k-means with MapRecude. k-means uses iterations by definition and Map Reduce doesn't support iterations [5][30], since each MapReduce job has a lot of I/O costs associated [30]. This suggests directly mapping k-means iterations to the MapReduce jobs would prove to have low performance. There are many solutions in the

literature that attempt to optimize k-means in a parallel environment running it in Hadoop's MapReduce in a variety of ways. Many works' goals are to minimize the execution time while maximizing cluster quality [13].

In the literature there are many solutions for implementing the k-means algorithm in a parallel environment using Map Reduce such as [37], and [38]. [10] is a survey on k-means clustering using MapReduce for Big Data. [32] uses genetic algorithm steps. [27] improves serial Two-Phase K-means using Incremental k-means algorithm. [24] studies this problem with real-time time-series data. There are many approaches proposed in the literature, such as optimizing the initial centroids chosen, minimizing the number of k-means iterations, minimizing the number of distance calculations, optimizing hardware configuration, outlier removal, etc. [25] and [11] remove outliers in attempt to optimize k-means over MapReduce.

There are quite a few applications to parallel k-means over MapReduce. [28] proposes an aspect based summary generation solution by mining opinions. [29] designs a k-means over MapReduce algorithm and compares it to serial k-means for document datasets. [34] implements k-means over MapReduce for image pattern recognition.

2.6.1 Initial Centroid Optimization

The following literatures attempt to optimize the initial centroids chosen: [3], [23], [4], [33], [9], [11], [22], [31], [36], [26], and [12]. [4] optimizes the initial centroids using data dimentionality density. [9] varies the centroids and data to optimize k-means. [12] optimizes initial centroid choice using the PSO meta-heuristics, which improves cluster quality and execution time. [11] uses two approaches, a), removing outliers from the datasets, and b), automating the initial centroid selection. [22] uses Min-Max normalization technique to choose better initial centroids, which requires assigning weights/priority to attributes of the dataset. The work done by [31] achieves better accuracy than the traditional k-means by taking averages of the dataset for better selecting the initial centroids. [36] compares their algorithm, Adaptively Disperse Centroids K-means Algorithm to Mahout. [26] introduces a preprocessing phase to compute the initial centroids, and then focuses on evaluation of cluster quality using data preprocessing, clustering, and pattern recognition.

2.6.2 Minimizing k-means Iterations

[18] introduces a preprocessing stage to k-means over MapRecude using k-d tree, to allow completion of the k-means algorithm in one MapReduce job. This work shows that with the same centroid configuration, their approach is faster and produces similar cluster quality compared to other literatures. [21] aims to optimize the execution time while keeping 80% accuracy of clusters by reducing iterations. [23] demonstrates (via simulation) that their work reduces the number of k-means iterations and increases the speed of the iterations. This works does so by analysing the dataset's distribution for initial centroid selection, and dynamically chooses between the Euclidean distance and Manhattan distance algorithms for comparing data element's distances. [30] proposes a single-pass MapReduce job for parallelizing k-means, called mrk-means. This work uses re-clustering and increases cluster quality. [33] automatically determines the number of clusters that will be generated, and only requires one MapReduce job, minimizing I/O cost to the file system.

2.6.3 Minimizing Number of Distance Computations

[6] reduces the number of distance computations performed, and achieve the same results. [5] reduces the distance computations using triangle inequality by using Extended Vector and Bounds Files. They compare both the Extended Vector and Bound Files approaches together. [8] carefully designs the Mapper and Reducer. Similarly, [7] attempts to reduce the number of reads and write to disk by the Mapper and Reducer. [35] reduces the number of iterations up to 30% and keeps up to 98% accuracy.

2.6.4 Optimiznig Hardware Configuration in Hadoop Environment

[15] explores the use of CPUs and GPUs using OpenCL to optimize k-means using MapReduce. [17] analyses the performance when adjusting processor micro-architecture parameters. [20] validates the importance of k-means over MapReduce by conducting experiments and varying the number of nodes in Hadoop environment.

2.7 PKMeans

This algorithm is proposed by [38]. It runs K-means in parallel using MapReduce, by running the point labeling and cluster center computations in MapReduce jobs, until the centroids converge.

Mapping Phase PKMeans uses the map phase to label the points to their nearest centroid, which can use any number of mappers. The input to the mappers is the dataset of points and the centroids for the current iteration (the initial centroids are given to the first MapReduce job). The output of the mapping phase is the labeled dataset, such that each data point is labeled with the id of the nearest centroid.

Reduce Phase Let k be the number of initial centroids. At each iteration PKMeans has a reducer, for each cluster, which computes the new centroid centers for its cluster. The input to the reduce phase is the labeled data points from the map phase and the centroids for the current K-means iteration. The output of the reduce phase is the set of new cluster centers (centroids).

2.8 IPKMeans

This algorithm is proposed by [18]. IPKMeans attempts to improve the execution time of the PKMeans algorithm. They compare the execution time and sum of squares error (SSE) between IPKMeans and PKMeans. The SSE is the sum of errors for each point, which is the distance from the target point to the actual point [19]. In this case the error is the distance from a point to its cluster center. IPKMeans requires a pre-processing phase and a post-processing, requiring multiple MapReduce jobs. They implement K-means in parallel using MapReduce in three phases, namely: the data partitioning phase, the parallel K-means phase, and the centroid merging phase.

2.8.1 Phase 1 - Data Partitioning

This phase builds a KDTree to partition the data set into subregions. A subregion is the set of points found in a KDTree leaf. Once the subregions are created, IPKMeans labels points from *1* to R, where R is the number of reducers. After labeling each subregion, subgroups are formed as a results. A subgroup contains all points with the same label. This data

partitioning scheme attempts to keep each subgroup representative of the original dataset's distribution. The input to this phase is the dataset, such that each point is labeled with a subregion id, and the output of this phase are the points labeled with a subgroup id. This phase is done in O(log$N$) MapReduce jobs.

2.8.2 Phase 2 - Parallel K-means

This IPKMeans phase runs R instances of K-means in parallel, on each of the subgroups obtained from phase 1 (see section 2.8.1). Since K-means finds k centroids, this phase will produce $R*K$ centroids found. This centroid set must be trimmed to only k centroids (see phase 3 section 2.8.3). The map phase organizes each data point by subgroup. The input of the map phase is the dataset labeled by subgroup and the output are the points organized by subgroup. The reduce phase runs K-means on each of the subgroups. The output of the reduce phase are the $R*K$ resulting centroids.

2.8.3 Phase 3 - Centroid Merging

This phase can be done using one machine, since there are only $R*K$ resulting centroids, obtained from phase 2 (see section 2.8.2). The centroids must be processed to find the most central centroid set. They refer to a central centroid set as set with the least average SSE. The input to this phase are many centroid sets each with their average SSE. The output of this phase are the k centroids found, the output of the IPKMeans algorithm.

3 Problem Statement

This work verifies the work done in [18], since there are a few points they make that raise question. They claim to propose IPKMeans, which outperforms the solution proposed in [38], PKMeans. They make a strong claim that they solve K-means using a single MapReduce job using IPKMeans, while PKMeans requires many MapReduce jobs. This claim is partially true. In a single MapReduce job K-means is executed in parallel, but it isn't completely solved using only one MapReduce job. They don't make it clear that phase 1 of IPKMeans (see section 2.8.1) requires O(log$N$) MapReduce jobs.

Another cause for evaluating their work is their experiments are done using a single machine that ran a multi-threaded Hadoop environment. Hadoop by nature runs in a multi-node environment, so their experiment results may have sources of error. The motivation for verifying [18]'s work, is to see if their claims hold when tested in an actual Hadoop cluster.

Some of their experiments lack test cases with large datasets. Since MapReduce is used in the context of Big Data, their experiments should hold when stressed with large datasets, but they only compare IPKMeans to PKMeans with a dataset of 3000 points. It isn't clear that their algorithm improves PKMeans when ran in a Big Data context.

4 Hadoop Cluster Configuration

The Hadoop cluster used to run the experiments discussed in this paper was configured using Carleton University's OpenStack cloud. OpenStack is an open source software tool to create clouds[2], and was used to create virtual machines (VMs) of Hadoop nodes to create a virtual Hadoop cluster. The VMs of the cluster were Bitnami VMs (obtained from [1]),

which are pre-configured Linux VMs with Hadoop pre-installed. The Hadoop cluster used to perform the experiments contained 10 Hadoop nodes, which were configured as follows (see Figure 1):

1. **Master Node**: implements the Name Node service for managing the HDFS nodes, and the Resource Manager service for scheduling MapReduce jobs on the slaves.
2. **Slave Nodes**: implements the Data Node service for HDFS storage, and the Node Manager service for running MapReduce jobs
3. **Client Node**: implements Hadoop Hive service for deploying Hadoop jobs, and the Secondary Name Node service as a backup Name Node service
4. **Service Node**: implements the Application Timeline Server to keep track of jobs

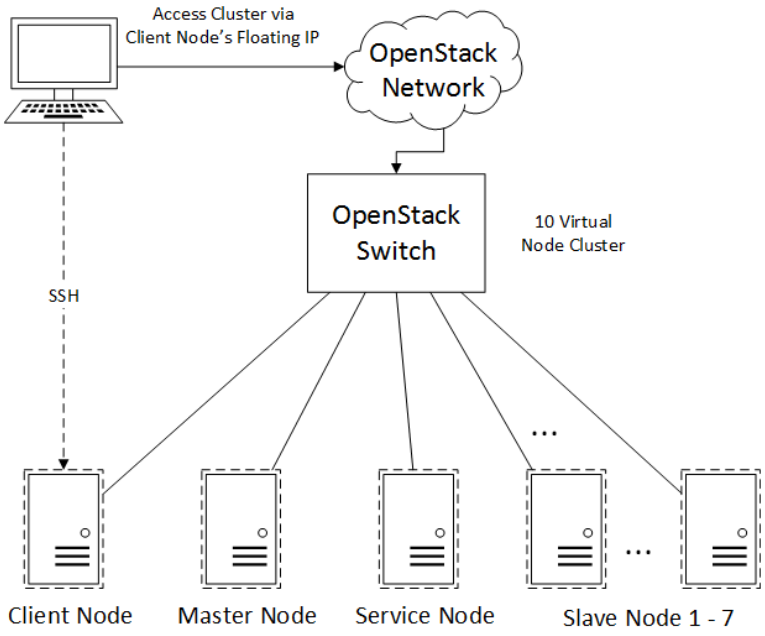

Figure 1: Hadoop cluster OpenStack virtual network diagram.

4.1 Hadoop Node Roles

The Client Node was used as the job deployment VM. By configuring OpenStack, SSH was enabled for accessing only the Client Node. The virtual network was configured such that the Client Node was the Hadoop cluster's access point, and the rest of the nodes shown in Figure 1 were only accessible from the Client Node. In total there were seven Slave Nodes, which were each in charge of storing HDFS file segments and running MapReduce jobs.

In order to have the network communication between each node, they needed to exchange SSH key-pairs before the cluster to be used. This allowed Hadoop services to communicate with each other and run remote commands without the need of user interaction (a password prompt).

4.2 Environment

The VMs were spread out over multiple physical host machines. The Client Node and the second Slave Node were hosted on the same physical machine, which had an Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00GHz (dual 14-core, 56VCPU) CPU.

All the other Hadoop nodes of the cluster were spread out over five physical hosts machines. Each of these hosts had an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz (dual 10-core, 40VCPU) CPU. Each virtual Hadoop node in the cluster had 1 core, 2GB of RAM, and 16GB of disk space. For all slave nodes, the Java Virtual Machine instances for both the mappers and reducers were allocated with 512MB of RAM. The OpenStack environment was shared with many other OpenStack users of Carleton University.

5 Results

This work reproduces a subset of the experiments ran in [18]. For various experiment results the SSE and execution time for IPKMeans and PKMeans are compared.

5.1 Experimental Datasets

[18] use five sets of initial centroids and a set of 3000 2-dimensional points, which have 3 clusters generated using a Gaussian distribution (see Figure 2-a). The reproduced dataset is shown in Figure 2-b, which was used to run the experiments discussed in this paper.

5.1.1 Data Points

To reproduce the 3000-point dataset shown Figure 2-a, since [18] didn't mention the mean or variance of the clusters they generated, this work attempted to reproduce the dataset by estimating the means and variances visually. The three cluster distribution estimations are shown below in the expression 1, which take the form $c_i = \{[\mu_x, \mu_y], [\sigma_x, \sigma_y]\}$, where c_i is cluster i, μ_d is the mean of c_i's dimension d, and σ_d is the variance of c_i's dimension d.

$$\begin{aligned} c_1 &= \{[5, 4], [3, 3]\}, \\ c_2 &= \{[13, 4], [3, 3]\}, \\ c_3 &= \{[15, 15], [3, 3]\} \end{aligned} \tag{1}$$

It is worth noting that the estimation of variance was done poorly, since the points in Figure 2-a are more scattered than the points in Figure 2-b. This was only discovered after running all the experiments, and there wasn't enough time to redo all the experiments with datasets that were generated with a better estimated variance. This may be a source of error to the experimental results found.

The experiments ran in this paper used 6 different randomly generated datasets (based on a Gaussian distribution) of various dataset sizes, namely datasets of 3000, 6000, 12000, 21000, 84000, and 192000 points. Each dataset share the same distributions described in expression 1 above, the only difference when they were generated was their size. The datasets were generated only once, and were used in multiple experiments. The datasets will be referred to in the rest of the paper as d_i, where i is the id of the dataset. The dataset definitions are defined below in the expression 2:

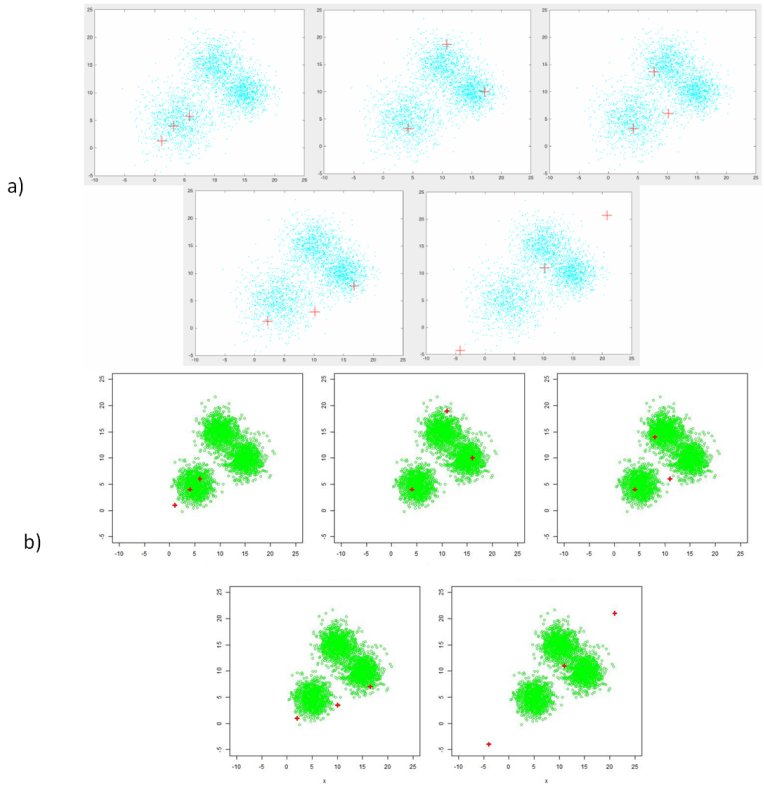

Figure 2: dataset with 3000 points and 5 groups of initial centroids marked by '+'. Chart a): IPKMeans dataset[18]. Chart b): reproduced dataset.

$$
\begin{aligned}
&| d_1 | = 3000 \\
&| d_2 | = 6000 \\
&| d_3 | = 12000 \\
&| d_4 | = 21000 \\
&| d_5 | = 84000 \\
&| d_6 | = 192000
\end{aligned}
\tag{2}
$$

5.1.2 Initial Centroids

The initial centroids, the red crosses in Figure 2, used in the experiments discussed in this paper were also estimated visually from Figure 2-a. They are described in the expression 3 below, where g_i is the initial centroid group i, which each contain three 2-dimensional points (the initial centroid coordinates).

$$
\begin{aligned}
g_1 &= \{(1,1),(4,4),(6,6)\} \\
g_2 &= \{(4,4),(11,19),(16,10)\} \\
g_3 &= \{(4,4),(8,14),(11,6)\} \\
g_4 &= \{(2,1),(10,3.5),(16.5,7)\} \\
g_5 &= \{(-4,-4),(11,11),(21,21)\}
\end{aligned} \tag{3}
$$

5.2 Experiments

The following sections discuss the experiments ran in this work and the results found. Some experiments will be comparing the results found in this work to the results found in [18]. The following notation will be used to differentiate between the experiments done in this work, and the experiments done in [18]: e_1 denotes the experiment(s) ran in this work, and e_2 denotes the experiment(s) (if it was conducted) ran in [18]. That is, the following sections will refer to e_1 as the experiment, or experiments, associated to the section they are mentioned in, and e_2 as that section's experiment, or experiments, ran in [18].

5.2.1 Reproducing IPKMeans Experiment — Initial Centroid Varying

This experiment varies the initial centroid groups, from g_1 to g_5, using the dataset d_1, and analyzes the comparison of execution time and SSE between PKMeans and IPKMeans.

Execution time The results of this experiment are displayed in Figure 3. The results of e_1 and e_2 are similar. For most initial centroid groups (g_1 to g_4), the execution time relationship between IPKMeans and PKMeans remains consistent (i.e. the results are the same). That is, for both results of e_1 and e_2 given some g_i, PKMeans is faster than IPKMeans, or IPKMeans is faster than PKMeans. For the experiment using g_1, IPKMeans outperforms PKMeans; using g_2, PKMeans outperforms IPKMeans; using g_3, PKMeans outperforms IPKMeans; and using g_4, IPKMeans outperforms PKMeans.

However, the experiment using g_5 is not consistent for e_1 and e_2. In e_2 IPKMeans outperforms PKMeans, and in e_1 PKMeans outperforms IPKMeans. This may be caused by a lucky K-means random centroid choice in one of the PKMeans iterations in e_1. That is, PKMeans in e_1 may have gotten lucky and chosen a random centroid that enabled quick convergence when one of its clusters was empty.

SSE The SSE results found for IPKMeans and PKMeans are quite similar for e_1 and e_2. The SSE of IPKMeans is generally higher (less accurate) than PKMeans, since IPKMeans attempts to optimize execution time at the expense of precision. See Table 1 below for the SSE e_1 results.

Initial Centroid Group	g_1	g_2	g_3	g_4	g_5
PKMeans	40698.09	17457.96	17457.93	40698.09	17457.93
IPKMeans	41107.72	17496.35	17496.35	40878.8	17496.35

Table 1: SSE PKMeans vs. IPKMeans, varying initial centroid group, dataset d_1, 7 reducers, and 10 hadoop nodes

Source of Error It is also worth mentioning that in e_1, for the experiments using g_4 and g_5, the average execution time for both IPKMeans and PKMeans is significantly higher

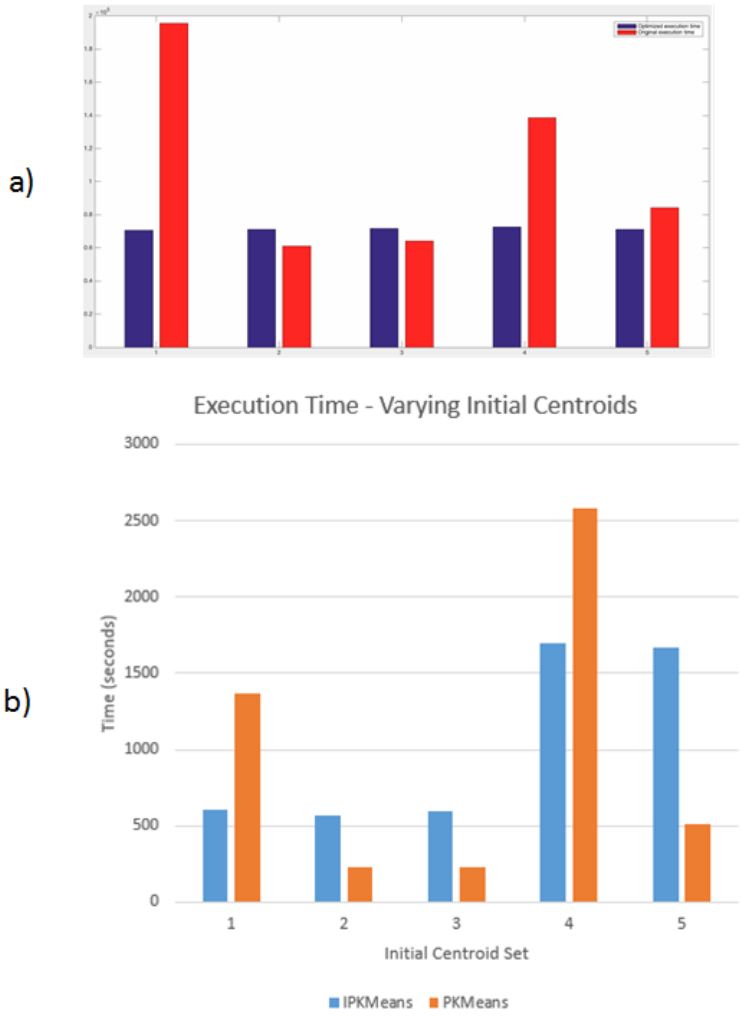

Figure 3: Varying initial centroid groups using dataset d_1. Chart a): results from e_2 [18]. Chart b): results from e_1 using 7 reducers and 10 Hadoop nodes.

than the experiments using g_1, g_2, and g_3 (see Figure 3-b). This may be due to a caching performance issue with the Hadoop nodes. The experiments done for initial centroid groups g_1, g_2, and g_3 were done 2 weeks before g_4 and g_5. The cluster had been running for quite a while in-between these experiments, and this may have created a caching issue in the Hadoop nodes. There were network connection issues during the experiments, that caused the Hadoop node communication to fail. The work around involved occasionally using the linux command 'dhclient', but this created an increasing number of persistent processes that ran in the background, potentially stealing resources from the Hadoop nodes.

Another possibility for the differences in execution times could be the Hadoop nodes were VMs of OpenStack. The VMs shared the host machine's hardware, and therefore other users may have also had VMs sharing the same hardware resources that the Hadoop nodes were using. In that case, other users may have been running experiments concurrently with the experiments of this paper, slowing down the execution time of e_1 experiments.

5.2.2 Varying Dataset Size

This experiment varies the dataset sizes (from d_2 to d_6), using initial centroid group g_1, and compares the execution time and SSE between IPKMeans and PKMeans. [18] doesn't compare IPKMeans to PKMeans when using greater dataset sizes.

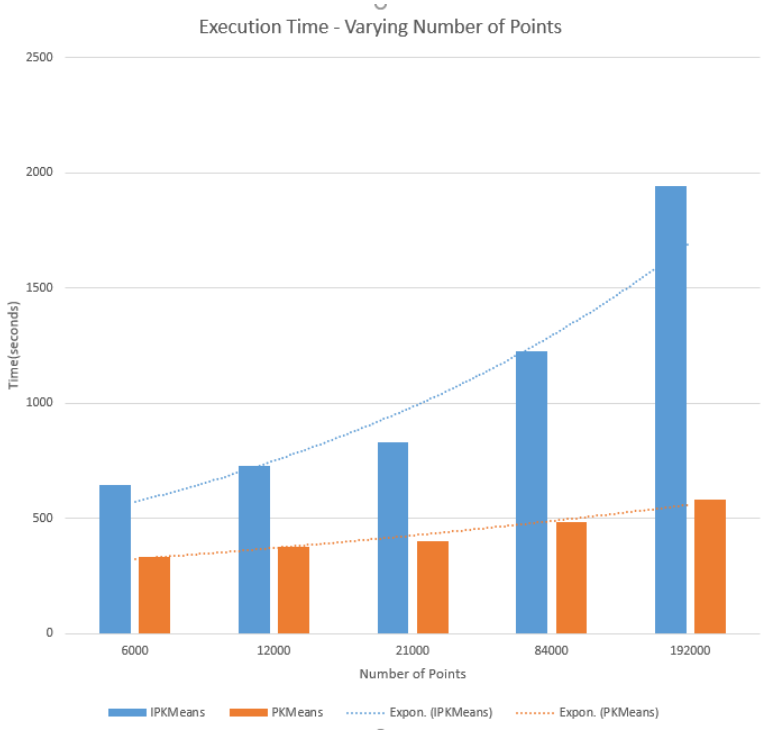

Figure 4: Varying dataset size, initial centroids group is g_1, 7 reducers, and 10 virtual hadoop nodes.

Execution time The results of e_1 are displayed in Figure 4, suggest that IPKMeans is much slower than PKMeans as datasets become large. IPKMeans' execution time increases exponentially as the dataset size increases, while PKMeans' execution time increases linearly. This experiment provides meaningful insight. When g_1 was used in the experiments discussed in section 5.2.1, IPKMeans outperforms PKMeans, but with many more points PKMeans outperforms IPKMeans in e_1.

SSE The SSE of PKMeans outperforms IPKMeans slightly (see Table 2 below), as expected, and the SSE of both PKMeans and IPKMeans increase exponentially as the dataset sizes increase.

Dataset	d_2	d_3	d_4	d_5	d_6
Size	6000	12000	21000	84000	192000
PKMeans	34962.71	70563.29	123944.57	500175.87	1516701.15
IPKMeans	34999.86	70568.12	123953.05	668659.23	1139632.76

Table 2: SSE PKMeans vs. IPKMeans, varying dataset size, initial centroid set g_1, 7 reducers, 10 hadoop nodes

Source of Error

The reason IPKMeans was completely outperformed by PKMeans (in execution time) as the dataset sizes increased may be because of the distribution of points. For example, if the data points were very sparse or if the data point dimensions were high, PKMeans may have struggled compared to IPKMeans.

5.2.3 Varying Initial Centroid

By analyzing the results of the experiments discussed in section 5.2.2, another initial centroid set may have had IPKMeans outperform PKMeans even with a large dataset. Upon examining the charts in Figure 3, it shows that IPKMeans and PKMeans outperform each other depending on the initial centroid set chosen. This experiment, e_1, reproduces the experiments mentioned in section 5.2.1, but it uses the dataset d_5 instead of d_1. e_1 was designed to answer the following question: is the initial centroid set responsible for dictating which algorithm outperforms the other when ran on large datasets?

Execution time Figure 5 illustrates the results of e_1, which suggest the initial centroid's have no effect on whether PKMeans outperforms IPKMeans when used on a large dataset. PKMeans is approximately three times faster than IPKMeans. These results are interesting since MapReduce is useful in the context of BigData, and if IPKMeans can only outperform PKMeans on small datasets, then IPKMeans doesn't improve PKMeans. With small datasets, it may faster to simply run K-means serially which will most likely outperform both PKMeans and IPKMeans.

By observing Figure 7 in the Appendix, which separates Figure 5 into two graphs, it is shown that the initial centroids are still affecting the performance of both IPKMeans and PKMeans. They greatly affect the execution time of both algorithms even when large datasets are used. The initial centroid set g_4 is the best for both algorithms when the dataset d_5 is used.

SSE The SSE of PKMeans outperforms IPKMeans in most cases (see Table 3 below), as expected. What is interesting is when g_4 is used, IPKMeans outperforms PKMeans. This could be the case for example, because PKMeans converged to a local SSE minimum centroid set, while IPKMeans found a global SSE minimum centroid set, since IPKMeans has many centroids to choose from, from the Parallel K-means phase (see section 2.8.2).

Initial Centroid Group	g_1	g_2	g_3	g_4	g_5
PKMeans	500175.87	500175.87	668642.8	665060.06	665060.06
IPKMeans	668659.23	500188.39	666841.75	500188.39	665252.59

Table 3: Experiment e_1: SSE PKMeans vs. IPKMeans while, initial centroid group, dataset d_5, 7 reducers, and 10 hadoop nodes

Source of Error

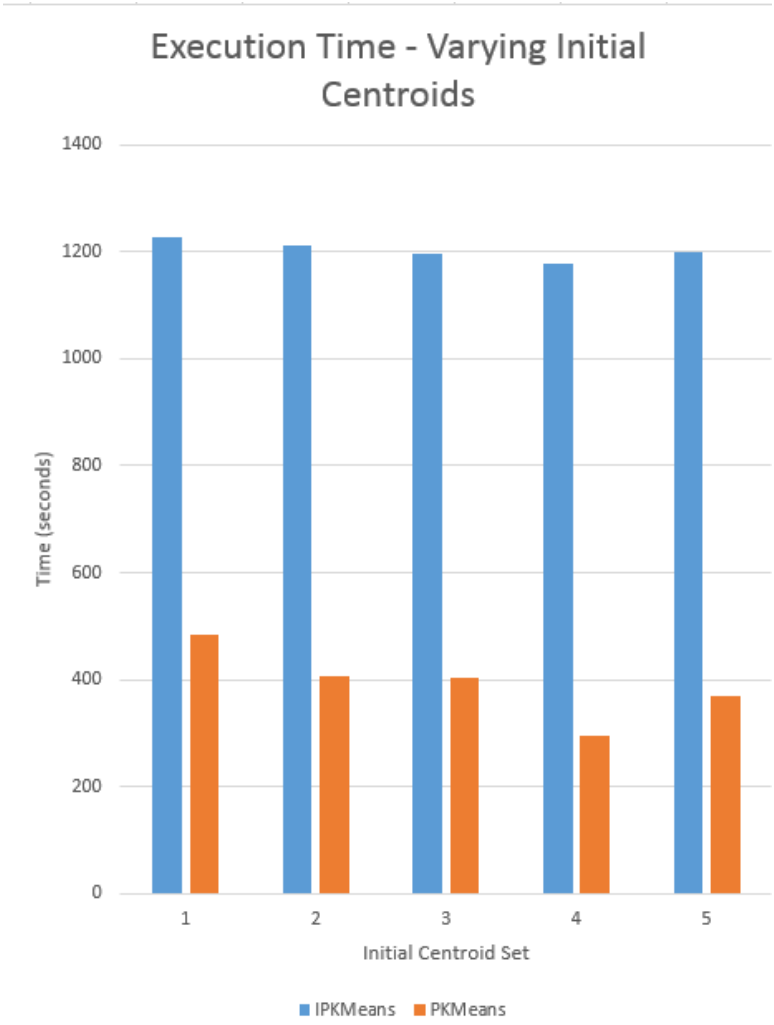

Figure 5: Varying initial centroid groups, dataset d_5, 7 reducers, and 10 virtual Hadoop nodes.

The sources of error may be caused by the same sources of error discussed in section 5.2.2.

5.2.4 Varying Number of Reducers and Slave Nodes

In this experiment, d_1 is the dataset used and the initial centroid group g_1 is used. The number of reducers and Hadoop nodes were varied, and the resulting SSE and execution time between IPKMeans and PKMeans are analyzed. The number of slave nodes is increased from two to seven.

The results shown in Figure 6 below and Figure 9 in the Appendix contradict the results found in [18]. When they ran the same experiment, the execution time decreased and the SSE increased as the number of reducers increased. In the results of e_1, the execution time increases and the SSE generally decreases as the number of reducers increase, but as the number of reducers becomes large the SSE increases. This may be the case since e_1 was

run in an actual Hadoop environment, while e_2 were ran on a single machine.

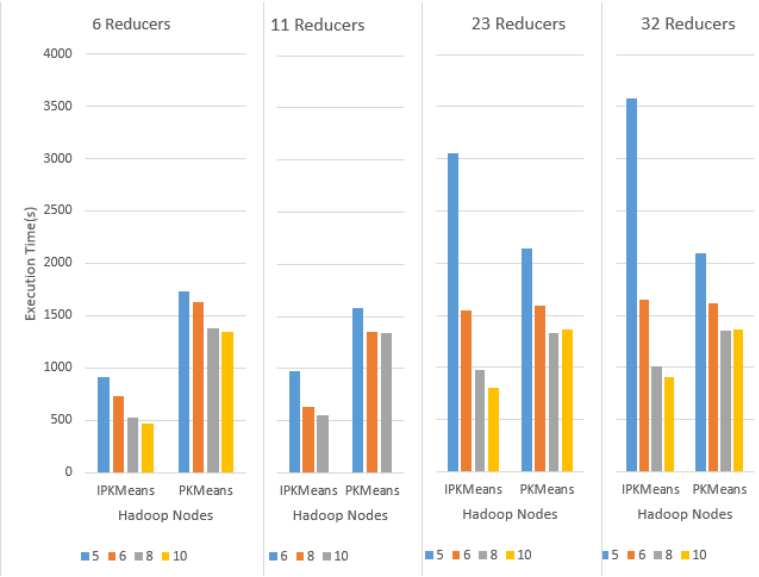

Figure 6: Execution time analysis when varying number of reducers and Hadoop nodes, using initial centroids group g_1 and dataset d_1

Execution time Illustrated in the Figure 6, as the number of nodes increase in the cluster, the execution time decreases for both IPKMeans and PKMeans. IPKMeans benefits more from the number of nodes, most likely because it can take advantage of many reducers in parallel, while PKMeans is limited to the k reducers in parallel. By looking at the case of 32 reducers for IPKMeans in Figure 6, increasing the number of Hadoop nodes from 5 to 6 speeds up IPKMeans by a factor of 2. PKMeans doesn't have a case where the speed up is a factor on 2. This suggests that as the number of Hadoop nodes increase, IPKMeans will benefit more than PKMeans since a huge number of reducers can be used by IPKMeans. PKMeans can still make use of the number Hadoop nodes by increasing its number of mappers.

IPKMeans does poorly compared to PKMeans with few Hadoop nodes as the number of reducers increase, but when there are many Hadoop nodes IPKMeans outperforms PKMeans. This is interesting, since it suggests that when datasets become huge, adjusting the configuration of Hadoop parameters (number of nodes, reducers, etc.) could enable IPKMeans to outperform PKMeans, since it appears PKMeans' speed-up converges more more quickly than IPKMeans' speed-up as the number of Hadoop nodes increase. It would be interesting to see if IPKMeans could outperform PKMeans in the experiments discussed in the sections 5.2.2 and 5.2.3, by increasing the cluster size and varying the number of reducers.

SSE The PKMeans' SSE didn't change much as reducers increased, since for PKMeans the number of reducers are fixed to k, where k is the number of clusters.

Illustrated in the Figure 9, the SSE of IPKMeans lowers as the number of reducers increases. A local SSE minimum appears for a certain number of reducers depending on the number of Hadoop nodes. As the number of reducers increase passed this optimal number, the SSE starts increasing. This suggests there exists an optimal number of reducers given

the number of Hadoop nodes in a cluster to minimize the SSE for IPKMeans. A better example of this can be shown in Figure 8, where 11 reducers is the optimal number of reducers for minimizing the SSE when using the dataset d_6 and using g_1.

Source of Error

The results of e_1 may be quite different if e_1 were conducted using each of the initial centroid groups instead of only using g_1.

6 Conclusions

Most of the claims made in [18] about their results were found to be correct, when their experiments were reproduced in this work. IPKMeans does outperform PKMeans depending on the initial centroids chosen when both algorithms are run on a normally distributed 3-cluster dataset with 3000 2-dimensional points. Some results found by [18] were not reproducible by the experiments run in this work. They found the SSE increases as the number of reducers increase, but this work has found that generally the SSE decreases as the number of reducers increase. However, when the reducer count becomes large the SSE does increase. This may be the case since [18] used a multi-threaded single machine Hadoop environment, while this work used a multi-node Hadoop environment to run experiments. As stated by [18], the SSE of IPKMeans tends to be worse compared to PKMeans for almost all experiments run in this paper.

This work has also ran new experiments. IPKMeans was to found to be outperformed (in execution time) by PKMeans with datasets larger than 3000 points. [18] ran experiments on IPKMeans with 15000 points, but they didn't include PKMeans in these experiments. With a small Hadoop cluster PKMeans outperforms IPKMeans, but some experimental results suggest IPKMeans could outperform PKMeans if the Hadoop cluster was large enough and the Hadoop parameters were configured properly. IPKMeans can really take advantage of the number of nodes in a cluster by increasing the number of reducers used to solve the problem, while PKMeans is limited to k reducers, where k is the number of centroids.

The work done in this paper hasn't fully explored the performance of PKMeans vs. IPKMeans. The future research to be made could be to further compare IPKMeans to PKMeans with a variety of datasets. Different datasets could be used by varying the number of clusters, distribution of points, point dimension, and number of points. It would also be interesting to run the experiments done in this work with a large Haddoop cluster to explore the effect of reducers vs. number of nodes in more detail. The number of centroids could also be varied, and many more initial centroid sets could be used. The datasets used in this paper could also have been generated to more closely to resemble the datasets used in [18].

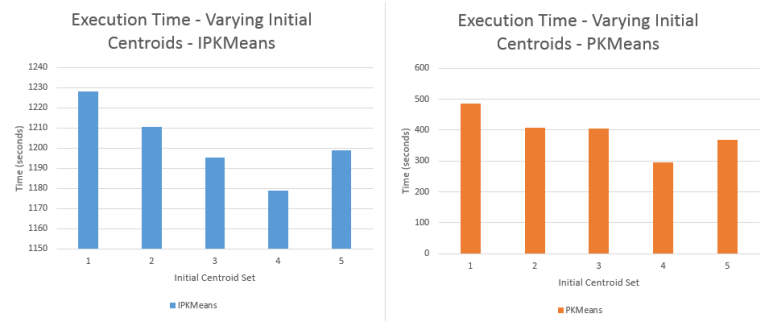

Figure 7: Varying initial centroids group, dataset d_5, 7 reducers, and 10 virtual hadoop nodes, a closer look at the effects on IPKMeans and PKMeans

7 Appendix

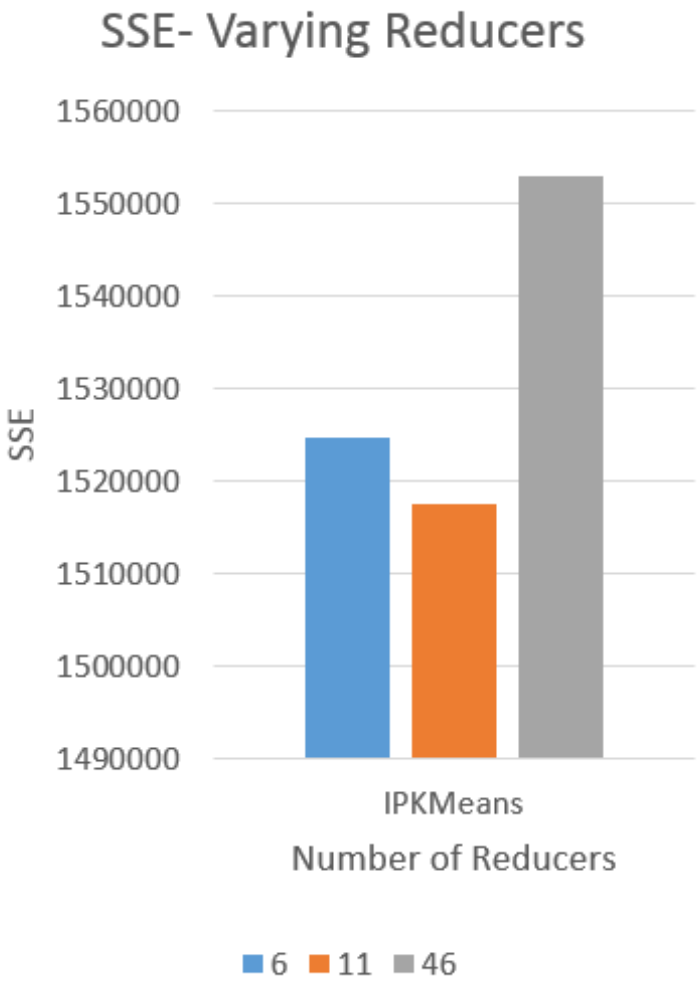

Figure 8: SSE analysis when varying number of reducers and with 10 hadoop nodes, initial centroids group is g_1, 192000 points

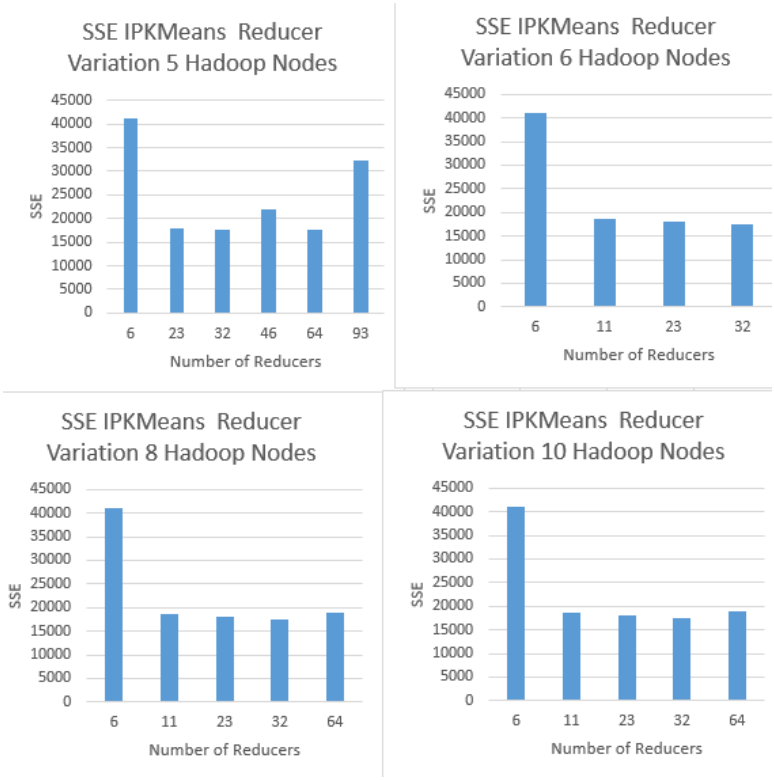

Figure 9: SSE analysis when varying number of reducers and hadoop nodes, initial centroids group is g_1, 3000 points

References

[1] Bitnami hadoop stack virtual machines. https://bitnami.com/stack/hadoop/virtual-machine. Accessed: 2018-11-26.

[2] Open source software for creating private and public clouds. https://www.openstack.org/. Accessed: 2018-11-26.

[3] Nadeem Akthar, Mohd Vasim Ahamad, and Shahbaaz Ahmad. Mapreduce model of improved k-means clustering algorithm using hadoop mapreduce. In *Computational Intelligence & Communication Technology (CICT), 2016 Second International Conference on*, pages 192–198. IEEE, 2016.

[4] Nadeem Akthar, Mohd Vasim Ahamad, and Shahbaz Khan. Clustering on big data using hadoop mapreduce. In *Computational Intelligence and Communication Networks (CICN), 2015 International Conference on*, pages 789–795. IEEE, 2015.

[5] Sami Al Ghamdi and Giuseppe Di Fatta. Efficient parallel k-means on mapreduce using triangle inequality. In *Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence & Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress*

(DASC/PiCom/DataCom/CyberSciTech), 2017 IEEE 15th Intl, pages 985–992. IEEE, 2017.

[6] Sami Al Ghamdi, Giuseppe Di Fatta, and Frederic Stahl. Optimisation techniques for parallel k-means on mapreduce. In *International Conference on Internet and Distributed Computing Systems*, pages 193–200. Springer, 2015.

[7] Prajesh P Anchalia. Improved mapreduce k-means clustering algorithm with combiner. In *Computer Modelling and Simulation (UKSim), 2014 UKSim-AMSS 16th International Conference on*, pages 386–391. IEEE, 2014.

[8] Prajesh P Anchalia, Anjan K Koundinya, and NK Srinath. Mapreduce design of k-means clustering algorithm. In *Information Science and Applications (ICISA), 2013 International Conference on*, pages 1–5. IEEE, 2013.

[9] Soumyendu Sekhar Bandyopadhyay, Anup Kumar Halder, Piyali Chatterjee, Mita Nasipuri, and Subhadip Basu. Hdk-means: Hadoop based parallel k-means clustering for big data. In *Calcutta Conference (CALCON), 2017 IEEE*, pages 452–456. IEEE, 2017.

[10] Rujal D Bhandari and Dipak P Dabhi. Extensive survey on k-means clustering using mapreduce in datamining.

[11] Amira Boukhdhir, Oussama Lachiheb, and Mohamed Salah Gouider. An improved mapreduce design of kmeans for clustering very large datasets. In *Computer Systems and Applications (AICCSA), 2015 IEEE/ACS 12th International Conference of*, pages 1–6. IEEE, 2015.

[12] Abdelhak Bousbaci and Nadjet Kamel. A parallel sampling-pso-multi-core-k-means algorithm using mapreduce. In *Hybrid Intelligent Systems (HIS), 2014 14th International Conference on*, pages 129–134. IEEE, 2014.

[13] Abdelhak Bousbaci and Nadjet Kamel. Efficient data distribution and results merging for parallel data clustering in mapreduce environment. *Applied Intelligence*, pages 1–21, 2017.

[14] Osama A Doreswamy and BR Manjunatha. Scalable k-means algorithm using mapreduce technique for clustering big data.

[15] Sandip A Ganage and Dr RC Thool Heshsham Abdul Basit. Heterogeneous computing based k-means clustering using hadoop-mapreduce framework. *International Journal of Advanced Research In Computer Science and Software Engineering*, 3(6), 2013.

[16] Bharath Kumar Gowru and Pavani Potnuri. Parallel two phase k-means based on mapreduce. *International Journal of Advance Research in Computer Science and Management Studies*, 22:3–11, 2015.

[17] Joseph Issa. Performance characterization and analysis for hadoop k-means iteration. *Journal of Cloud Computing*, 5(1):3, 2016.

[18] Shikai Jin, Yuxuan Cui, and Chunli Yu. A new parallelization method for k-means. *arXiv preprint arXiv:1608.06347*, 2016.

[19] John D Kelleher, Brian Mac Namee, and Aoife DArcy. Fundamentals of machine learning for predictive analytics, 2015.

[20] Amresh Kumar, M Kiran, and BR Prathap. Verification and validation of mapreduce program model for parallel k-means algorithm on hadoop cluster. In *Computing, Communications and Networking Technologies (ICCCNT), 2013 Fourth International Conference on*, pages 1–8. IEEE, 2013.

[21] Oussama Lachiheb, Mohamed Salah Gouider, and Lamjed Ben Said. An improved mapreduce design of kmeans with iteration reducing for clustering stock exchange very large datasets. In *Semantics, Knowledge and Grids (SKG), 2015 11th International Conference on*, pages 252–255. IEEE, 2015.

[22] K Gaja Lakshmi and D Prabha. Clustering big data using normalization based k-means algorithm. 2016.

[23] Qing Liao, Fan Yang, and Jingming Zhao. An improved parallel k-means clustering algorithm with mapreduce. In *Communication Technology (ICCT), 2013 15th IEEE International Conference on*, pages 764–768. IEEE, 2013.

[24] Yongzheng Lin, Kun Ma, Runyuan Sun, and Ajith Abraham. Toward a mapreduce-based k-means method for multi-dimensional time serial data clustering. In *International Conference on Intelligent Systems Design and Applications*, pages 816–825. Springer, 2017.

[25] Li Ma, Lei Gu, Bo Li, Yue Ma, and Jin Wang. An improved k-means algorithm based on mapreduce and grid. *International Journal of Grid & Distributed Computing*, 8(1), 2015.

[26] Veronica S Moertini and Liptia Venica. Enhancing parallel k-means using map reduce for discovering knowledge from big data. In *Cloud Computing and Big Data Analysis (ICCCBDA), 2016 IEEE International Conference on*, pages 81–87. IEEE, 2016.

[27] Cuong Duc Nguyen, Dung Tien Nguyen, and Van-Hau Pham. Parallel two-phase k-means. In *International Conference on Computational Science and Its Applications*, pages 224–231. Springer, 2013.

[28] V Priya and K Umamaheswari. Ensemble based parallel k means using map reduce for aspect based summarization. In *Proceedings of the International Conference on Informatics and Analytics*, page 26. ACM, 2016.

[29] Tanvir Habib Sardar and Zahid Ansari. An analysis of mapreduce efficiency in document clustering using parallel k-means algorithm. *Future Computing and Informatics Journal*, 2018.

[30] Saeed Shahrivari and Saeed Jalili. Single-pass and linear-time k-means clustering based on mapreduce. *Information Systems*, 60:1–12, 2016.

[31] Rajashree Shettar and Bhimasen V Purohit. A mapreduce framework to implement enhanced k-means algorithm. In *Applied and Theoretical Computing and Communication Technology (iCATccT), 2015 International Conference on*, pages 361–363. IEEE, 2015.

[32] Puja Shrivastava, Laxman Sahoo, Manjusha Pandey, and Sandeep Agrawal. Akmaugmentation of k-means clustering algorithm for big data. In *Intelligent Engineering Informatics*, pages 103–109. Springer, 2018.

[33] Ankita Sinha and Prasanta K Jana. A novel mapreduce based k-means clustering. In *Proceedings of the First International Conference on Intelligent Computing and Communication*, pages 247–255. Springer, 2017.

[34] Anil R Surve and Nilesh S Paddune. A survey on hadoop assisted k-means clustering of hefty volume images. *International Journal on Computer Science & Engineering*, 6(3):113–117, 2014.

[35] Duong Van Hieu and Phayung Meesad. Fast k-means clustering for very large datasets based on mapreduce combined with a new cutting method. In *Knowledge and Systems Engineering*, pages 287–298. Springer, 2015.

[36] Bin Wang, Zheng Lv, Jinwei Zhao, Xiaofan Wang, and Tong Zhang. An adaptively disperse centroids k-means algorithm based on mapreduce model. In *Computational Intelligence and Security (CIS), 2016 12th International Conference on*, pages 142–146. IEEE, 2016.

[37] Hongbo Xu, Nianmin Yao, Qilong Han, and Haiwei Pan. Parallel implementation of k-means clustering algorithm based on mapreduce computing model of hadoop. *Metallurgical & Mining Industry*, (4), 2015.

[38] Weizhong Zhao, Huifang Ma, and Qing He. Parallel k-means clustering based on mapreduce. In *IEEE International Conference on Cloud Computing*, pages 674–679. Springer, 2009.