# Parallel K-Means Clustering Algorithm on Map Reduce Framework

Seung-Hee Kim[1], Jun-Ki Min[2]

[1] *Department of IT Convergence Software Engineering, Korea University of Technology and Education, Republic of Korea, sh.kim@koreatech.ac.kr*

[2] *School of Computer Science and Engineering, Korea University of Technology and Education, Republic of Korea, jkmin@koreatech.ac.kr*

***Article Info***
***Volume 81***
***Page Number: 2280 - 2290***
***Publication Issue:***
***November-December 2019***

***Article History***
***Article Received:*** *5 March 2019*
***Revised:*** *18 May 2019*
***Accepted:*** *24 September 2019*
***Publication:*** *12 December 2019*

***Abstract***

Clustering is the algorithm for partitioning the points in a given data set into several groups. The goal of clustering is that the points in a group are similar while the dissimilar points are in the different groups. Among the diverse clustering algorithms, K-Means as a center based clustering algorithm is one of the most widely used algorithms. In this paper, we propose an efficient parallel K-mean algorithm, called MPKMeans, which utilizes the MapReduce framework for processing large volume data sets. In MPKMeans, for each center, we maintain the distance to the closest center of it and we check whether each point is needed to compute the distances to all center points by using the maintained minimal distances. Additionally, in contrast to the existing parallel algorithm, since we design MPKMeans composed of map phases only, we eliminate the overhead for conducting shuffle and reduce phases.

***Keywords**: Clustering, K-Means, MapReduce, BigData*

## 1. INTRODUCTION

Clustering is one of fundamental problem in the data analysis and understanding. The goal of data clustering is to split a set of multi-dimensional points into groups known as clusters such that the points belonging to the same group are as similar as possible whereas the dissimilar points are located in different clusters.

One of the most widely used data clustering methods is K-means algorithm [1] for unsupervised classification and and analysis for multivariate observations. Owing to its simplicity and its linearity of time complexity, K-means clustering algorithm is one of the most popular and efficient algorithm. It has been widely used in diverse applications such as such as recommendation [2, 3], data mining [4, 5], machine learning [6], pattern recognition [7].

As an unsupervised learning algorithm, K-Means algorithm is based on iterative and repetitive processing. The requirement of K-Means is to provide the number of clusters $k$ as an input parameter. Given a number of clusters $k$, K-Means algorithm partitions a $d$-dimensional data set $D$ ==$\{p_1, p_2, ..., p_n\}$ into $k$ clusters and computes the center of each cluster iteratively.

In each iteration, the computation of the distance from each point $p$ in $D$ to centers calculated in the previous iteration is the crux of K-Means in order to assign $p$ into the closest center which represents a cluster. Furthermore, at the initial iteration, $k$ centers are randomly generated since there are no centers computed with the data set $D$. Furthermore, in K-means algorithm, the results of one iteration are not stored to feed the next iteration. Each calculation is done on original dataset. Therefore, as the number *n* and dimension *d* of points increase, execution time of K-means algorithm becomes slow down.

Nowadays, the volume of data increases explosively due to the proliferation of internet and smart devices and thus classical tool and systems are not applicable for bigdata. Therefore, the traditional serial K-means algorithm running on a single machine becomes computationally infeasible. For such data-intensive applications, the MapReduce [8] has recently attracted a lot of attention. As a distributed and parallel data processing framework, MapReduce provides a programming model that allows easy development of scalable parallel applications to process big data on large clusters of commodity machines. Google's MapReduce or its open-source equivalent Hadoop [9] is a powerful tool for building such applications.

In this paper, we propose an efficient parallel K-means clustering algorithm, called MPKMeans (abbreviated from Mapreduce based Parallel K-Means), running on the MapReduce framework.

Our proposed MPKMeans algorithm calculates *k* centers of clusters by running *t* rounds. In contrast to previous MapReduce algorithms [10,11,12], each round of MPKMeans consists of a single map phase. Thus, we can alleviate the overhead for synchronization, data grouping and network transmission during the shuffle and reduce phases. Furthermore, to reduce the computational overhead for calculating the distance from each point to each center, we devise a method in which we effectively identify the points whose closet center is not changed.

The rest of this paper is organized as follows. In Section 2, briefly explain the traditional K-means algorithm and MapReduce. Section 3 provides the summary of related work. In Section 4, we present the details of our proposed parallel MPKMeans algorithm. We demonstrate the efficiency of our developed algorithm compared with previous parallel K-Means algorithm in Section 5. The research is summarized in Section 6.

## 2. PRELIMINARY

Since our work is based on parallel data clustering, we first explain the behavior of the traditional K-means algorithm and next present the feature of the MapReduce framework.

### 1.1 K-means algorithm

In 1967, MacQueen [1] developed K-Means clustering algorithm for unsupervised classification and analysis for multivariate observations. Since K-means clustering technique has long and rich history, there are many variants. Among the diverse variants [13,14,15,16,18] of serial K-means clustering algorithm, Lloyd's algorithm [13] is regarded as a typical K-Means algorithm. Figure 1 shows the pseudo code of Lloyd's algorithm.

```
Procedure K-Means($D$, $k$)
$D$: a set of $d$-dimensional data set
$k$: the number of clusters
1: initialize $k$ centers $K = \{c_1, c_2, ..., c_k\}$
2: repeat
3:   for each $p \in D$ do
4:     find closest center $c_i$ in $K$
5:     assign $p$ to the Cluster $C_i$
6:   update every $c_i \in K$ using the points in each $C_i$
7: until there are no changes in $K$
```

**Figure 1**: Pseudo code of Lloyd's algorithm

As shown in Figure 1, K-means algorithm takes a set of *d*-dimensional data set $D=\{p_1, p_2, ..., p_n\}$, and the number of clusters *k*. Initially, *k* centers are randomly generated (line 1 of K-Means in Figure 1). Let a set of *k* centers be $K = \{c_1, c_2, ..., c_k\}$. For each point *p* in *D*, the closest center $C_i$ among *k* centers is identified and assign *p* to the cluster $C_i$ represented by $C_i$ in the for loop (lines 3-5). After assigning every point *p* to the cluster $C_i$ whose center $C_i$ is closest among *k* centers $c_1$,

$c_2,..., c_k$, we compute the center of each cluster $C_i$ by using the points allocated in $C_i$ (line 6). Ideally, the main loop (lines 3-6) of K-means algorithm is repeated until the $k$ centers converges and there is no change (line 7). However, in practice, this condition may never be satisfied or its satisfaction may require massive iterations. Thus, in general, the main loop of the K-Means algorithm is usually bounded to a fixed number $t$.

Given a $d$-dimensional data set $D=\{p_1, p_2, ..., p_n\}$, since we have to compute the distance from each point $p$ in $D$ to every center in $K$, the time complexity of the main loop of K-means becomes O($dnk$) and since the maximum iteration number is $t$, the time complexity of K-means algorithm is O($tdnk$).

Although K-means algorithm is simple and shows the linear time complexity, it has the following shortcomings. The first one is that it is hard to know the proper cluster number $k$ for a given data set in advance. The second thing is the high dependency of clustering quality to the initial $k$ centers. The last is that the calculation of the distance between each $d$-dimensional point $p$ in $D$ and each center $c_i$ in $K$ in order to find the closest center for each point $p$ is the performance bottleneck of K-Means since the results of one iteration are not stored to feed the next iteration. In other words, each calculation is done on original dataset. In this work, we alleviate the last shortcoming by developing an efficient parallel K-means algorithm.

## 1.2 MapReduce

Inspired by the map and reduce primitives present in functional languages, Google developed the MapReduce [8] framework that enables the users to easily develop large scale distributed applications. *MapReduce* is a distributed as well as parallel processing model and execution environment in the shared-nothing clusters of commodity machines. Hadoop [9] is implemented in the Open Source community as the MapReduce framework. In Hadoop, using the Hadoop Distributed File System (HDFS), a large sized file is initially partitioned into several fragments, called chunk, and stored in several machines redundantly for reliability. The size of a chunk is typically 64 MB.

In MapReduce, a program consists of a map function and a reduce function which are user-defined functions. The associated implementation parallelizes large computations easily as each map function invocation is independent and uses re-execution as the primary mechanism of fault tolerance. Basically, the MapReduce framework consists of one *job tracker* and several *task trackers*. Each task tracker is running on a commodity machine, called *slave*. A slave processes data using a map function and a reduce function each of which is invoked by the task tracker. The job tracker running on a single machine, called *master*, takes responsibility for error detection, load balancing and so on.

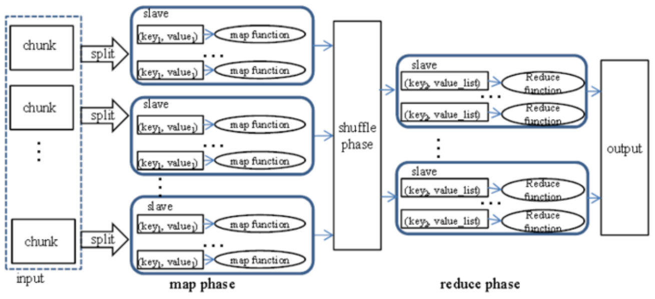

**Figure 2:** Data flow in MapReduce

Figure 2 illustrates the data flow in the MapReduce framework. Conceptually, the map and reduce functions implemented by the user have the following types:

$$map(key_1, value_1) \rightarrow list(key_2, value_2)$$

$$reduce(key_2, list(value_2)) \rightarrow (key_3, list(values_3))$$

In the MapReduce framework, the computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The data processing in MapReduce is composed of three phases: *map* phase, *shuffle* phase and *reduce* phase.

**Map phase:** In the map phase, a task tracker is called a *mapper*. A map function invoked by a mapper takes a key-value pair *(key1, value1)* as input, executes some computation and may output a set of intermediate key-value pairs *(key2, value2).* In addition, by applying a combine function, additional computation to intermediate results can be executed.

**Shuffle phase:** In this phase, intermediate key-value pairs are grouped with respect to the key *key2*. Thus, a reduce function to be executed in the next phase can obtain a list of values having the same key. Therefore, through this phase, each work is assigned the data lists as *(key2, list(value2)).*

**Reduce phase:** Finally, a reduce function with a key and a value list is invoked by a task tracker, called a *reducer*, at the reduce phase. Each reduce function executes computation for the value list and emits a key-value list pair *(key3, list(value3))* as a final result. The intermediate values are supplied to the reduce function via an iterator. This allows MapReduce to handle lists of values that are too large to fit in main memory.

Furthermore, each mapper can invoke a *setup* function before executing map functions and a *cleanup* function after executing all map functions. Similarly, each reducer also has the setup and cleanup functions.

## 3. RELATED WORK

In this section, we briefly present several variants of K-Means algorithm including parallel version of K-Means clustering.

To solve the issue of estimating the right number of clusters, G-means [15] and X-means[16] were proposed. To learn the integer *k* required in K-means, G-means [15] uses Gaussian mixture model. Starting with a small value of *k*, G-means splits the clusters that fail a test of spherical Gaussianity. Between each statistical test, the K-means algorithm is applied to refine the solution. In X-means [16], Bayesian information criterion (BIC) [17] is used to merge or split the clusters generated by K-means algorithm.

To alleviate the dependency of clustering quality to the initial *k* centers, K-means++ was proposed in [18]. The K-means++ is identical to the K-mean except the selection of initial *k* centers. K-means++ tries to select carefully the set of initial centers instead of random generation. Initially, a single center is randomly selected from a set of points *D* in K-means++. Let *dist(p)* denote the shortest distance from a data point *p* in *D* to the closest center among the centers which we have already chosen. Then, K-means++ chooses a point $p_i$ in *D* as another center with probability $dist(p_i)/\Sigma_{p\in D}\ dist(p)$ until *k* initial centers are generated. In other words, K-means++ tries to choose the points in *D* as initial centers which are far from each other. In [20], another method for initial center generation is proposed. In [20], K-means algorithm is conducted on *j* sample groups to obtain $j{\cdot}k$ centers. Then, K-means algorithm is applied again to the generated $j{\cdot}k$ centers to generate *k* centers.

For several years, to speed up K-means clustering, several parallel algorithms [10,11,19,12] have been proposed. However, some parallel K-Means algorithms [10,11] are a simple extension of a serial K-means algorithm. In [10,11], the cluster of every point *p* is identified by compute the distance from *p* to every center of each cluster at the map phase. Then, during the shuffle phase, the points belonging to the same cluster are grouped. Finally, new center of each cluster are calculated by using the points in the cluster at the reduce phase. This procedure are repeated for maximum number of iteration or every cluster’s center is not changed. In the whole process of iteration, since reducers in the MapReduce framework should fetch the intermediate data blocks from remote nodes, it will cause a large amount of communication overheads.

Although the parallel algorithm presented in [19] consists of a single MapReduce round, since it computes *k* centers approximately, it is not an actual K-means algorithm. In [12], to reduce the computational overhead, when a point *p* is an extreme point, the distance from *p* to a center is calculated by $L_1$ distance (i.e., manhattan distance) instead of $L_2$ distance (i.e., Euclidian distance). To decide whether p is an extreme point, kurtosis is used in [12]. However, since kurtosis is the statistical measure over the data set *D*, it is inapplicable to individual point. Thus, the algorithm in [12] is technically incorrect. Additionally, in [12] the large computational overhead for obtaining the farthest point from a center occurs.

In [21], to reduce the computational overhead of K-means clustering, a kd-tree is utilized. In this technique, the number of distance calculation is reduce by filtering a points which cannot belong to a cluster by traveling the kd-tree. However, in the MapReduce framework, there is no functionality provided for building and accessing such spatial indexes because it is difficult to provide efficient and scalable distributed indexes in multiple machines. Thus, it is hard to apply the technique proposed in [21] to the MapReduce framework.

## 4. MPKMEANS

In this section, we present the details of our proposed algorithm MPKMeans. Our proposed MPKMeans algorithm calculates *k* centers of clusters by running t rounds. In contrast to previous MapReduce algorithms [10,11,12], each round of MPKMeans consists of a single map phase. Thus, we can alleviate the overhead for synchronization, data grouping and network transmission during the shuffle and reduce phases. Furthermore, we devise a method in which the computational overhead by exploiting the minimum distance from each center ci to the other centers.

**Definition 1.** Given a center $c_i$ in a set of k centers $K = \{c_1, c_2, \ldots, c_k\}$, $cmdist_i = \min_{1 \le j \le k,\, j \ne i}$ distance($c_i$, $c_j$), where ($c_i$, $c_j$) is $\|c_i - c_j\|$. In other words, $cmdist_i$ denotes the minimum $L_2$ distance of the center $c_i$ to the other centers in *K*.

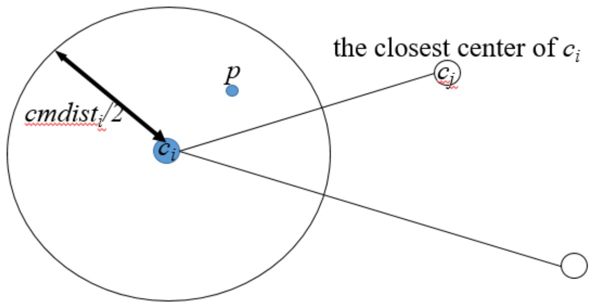

**Figure 3:** The relationship of a point p and *cmdisti*

Figure 3 illustrates that a point *p* within the half of $cmdist_i$ from a center $c_i$, $c_i$ is the closest center of *p* among $c_1, c_2, \ldots, c_k$. Let us assume that $c_j$ is the closest center of the center $c_i$ as shown in Figure 3. Then, when a point *p* is located within a half of cmdisti from $c_i$, *p* is closer to $c_i$ than $c_j$. Furthermore, since $c_j$ is the closest center of $c_i$, $c_i$ is the closest center of *p*. From the above observation, we devise the following lemma.

**Lemma 1**. Given a center $c_i$ in a set of *k* centers $K = \{c_1, c_2, \ldots, c_k\}$ and a point $p \in D$, if distance($p$, $c_i$) $\le cmdist_i/2$, $c_i$ is the closest center of *p* among $c_1, c_2, \ldots, c_k$.

**Proof.** When distance($p$, $c_i$) $\le cmdist_i/2$ is satisfied, 2distance($p$, $c_i$) $\le cmdist_i$. By Definition 1, since $cmdist_i$ is the distance from $c_i$ to its closest center, $cmdist_i \le$ distance($c_i$, $c_j$) always holds for the every centers $c_j$ ($\ne c_i$) in *K*. Thus, we have 2distance($p$, $c_i$) $\le$ distance($c_i$, $c_j$). Then, we get

$$\text{distance}(p, c_i) \le \text{distance}(c_i, c_j) - \text{distance}(p, c_i)\,. \quad (1)$$

In addition, by triangle inequality, we know distance($c_i$, $c_j$) ≤ distance($p$, $c_i$) + distance($p$, $c_j$). Then, we have

distance($c_i$, $c_j$) - distance($p$, $c_i$) ≤ distance($p$, $c_j$).     (2)

By using (1) and (2), we get distance($p$, $c_i$) ≤ distance($p$, $c_j$) for every center $c_j$ ($\neq c_i$) in $K$ if distance($p$, $c_i$) ≤ $cmdist_i/2$. In other words, $c_i$ is the closet center of $p$.♦

Given a cluster $C_i$ whose center $c_i$, if a point $p$ is located within $cmdist_i/2$, we do not need to calculate the distance from $p$ to other centers since $c_i$ is the closest center of $p$ by Lemma 1.

The pseudo code of our proposed algorithm MPKMeans is presented in Figure 4. As shown in Figure 4, MPKMeans takes $D$, $k$, $t$ which are the set of $d$-dimensional data, number of clusters and number of iterations, respectively. Initially, we generate a set of $k$ centers $K$ (line 1 of MPKMeans in Figure 4). We then calculate $cmdist_i$ for each center $c_i$ in $K$ by invoking compute_cmdist() and keep the all $cmdist_i$s into $M$ (line 2 of

```
Procedure MPKMeans(D, k, t)
D: a set of d-dimensial data set
k: the number of clusters
t: the number of iteration
1: initialize k centers K ={c_1, c_2,..., c_k}
2: M = compute_cmdist(K)
3: broadcast K, M
4: for itr = 1 to t do
5:     S= RunMapReduce(ClusterRound)
6:     K = compute_centers(S)
7:     M = compute_cmdist(K)
8:     broadcast K, M
```

**Figure 4**: Pseudo code of MPKMeans

MPKMeans in Figure 4). In other words, $M$ is {$cmdist_1$, $cmdist_2$, ..., $cmdist_k$}. To utilize the center set $K$ and minimum distance set $M$ in every machine, $K$ and $M$ are broadcast to every machine participated in the MapReduce framework. Then, MPKMeans executes the MapReduce algorithm ClusterRound for $t$ times and, at the end of each iteration, MPKMeans updates $K$ by invoking compute_centers procedure with the result of ClusterRound $S$ and recalculates all cmdistis in $M$ by using the newly computed centers (lines 4-7). Note that, to compute the new center of each cluster, the sum and number of points belonging to the cluster are required. And next, since $K$ and $M$ are required at the next round, $K$ and $M$ is broadcast repeatedly (line 8 in Figure 4).

Figure 5. Pseudo code of ClusterRound

```
ClusterRound.Mapper
ClusterRound.setup()
1: sums = {sum_1=0, sum_2=0, ..., sum_k=0}
2: counts = {count_1=0, count_2=0, ..., count_k=0}

ClusterRound.map(key, p)
key: null
p: a point in D
1: minDist = ∞, idx = -1, i =1
2: for i = 1 to k do
3:   if (distance(p, c_i) < minDist)
4:      idx = i
5:      if(dist(p, c_i) < cmdist_i /2) break;
6:      minDist = distance(p, c_i)
7: sum_idx = sum_idx+p
8: count_idx++

ClusterRound.cleanup()
1: for i = 1 to k do
2:     emit(null, <i, sum_i, count_i>)
```

**Figure 5**: Pseudo code of ClusterRound

Figure 5 illustrates pseudo code of ClusterRound invoked by MPKMeans. In contrast to the previous MapReduce algorithms, our proposed algorithm MPKMeans conducts the map phase only in each round.

As mentioned in Section 2, each mapper can invoke a *setup* function before executing map functions. In our algorithm, setup function is used for initializing the variables sums = {$sum_1$, $sum_2$, ..., $sum_k$} and counts = {$count_1$, $count_2$, ..., $count_k$} which keep the sum of coordinates of the points belonging to clusters and the numbers of points in clusters (lines 1-2 of ClusterRound.setup in Figure 5).

In the map function of ClusterRound in Figure 5, minDist keeps the current minimum distance from a point $p$ to a center $c_{idx}$. Thus, for each center $c_i$ with $1 \leq i \leq k$, we calculate distance($p$, $c_i$) and if distance($p$, $c_i$) is less than minDist, we keep the index $i$ in the variable $idx$ and update minDist by distance($p$, $c_i$) (lines 2-6 of ClusterRound.map in Figure 3). Furthermore, if distance($p$, $c_i$) $\leq cmdist_i/2$, we do not need to calculate the distances from $p$ to the other centers since $c_i$ is the closest center of $p$ by Lemma 1. Thus, when such a condition is satisfied, we exit the loop for finding the closest center (line 5). Finally, we accumulate the coordinates of $p$ in $\text{sum}_{idx}$ and increase $\text{count}_{idx}$ by 1 (lines 7-8 of ClusterRound.map in Figure 5) since they are sufficient to compute new center of the cluster $C_{idx}$ at line 6 in Figure 4.

After all points are processed by the mapper, the accumulated coordinate and count for each cluster are emitted by the cleanup function (lines 1-2 of ClusterRound.cleanup in Figure 5).

```
Procedure compute_centers($S$)
1: tss = {$ts_1$=0, $ts_2$=0, ..., $ts_k$=0}
2: tcs = {$tc_1$=0, $tc_2$=0, ..., $tc_k$=0
3: for each <$i$, $sum_i$, $count_i$> in $S$ do
4:     $ts_i = ts_i + sum_i$
5:     $tc_i = tc_i + count_i$
6: create $K = \{c_1, c_2, ..., c_k\}$ where $c_i = ts_i/tc_i$
7: return $K$
```

**Figure 6**: compute_centers procedure

Figure 6 shows the procedure compute_centers invoked by MPKMeans at line 6 in Figure 4. In the procedure compute_centers, he every $\text{sum}_i$ and $\text{count}_i$ for a cluster $C_i$ (with $1 \leq i \leq k$) emitted by every mapper are accumulated in $ts_i$ and $tc_i$, respectively (lines 3-5 in Figure 6). Then, the new center $c_i$ for each cluster $C_i$ is computed by dividing $ts_i$ by $tc_i$ and the set of newly computed centers $K$ are returned (lines 6-7 in Figure 6).

## 4. EXPERIMENTS

### 4.1 Experimental Environment

We show the efficiency of our proposed parallel algorithm MPKMeans by comparing with another parallel algorithm PKMeans [10].

To perform the experiments, we ran our implemented algorithms on a cluster of 36 commodity machines. One of machines acts as the master and the others act as slaves. The master has 3.1GHz Intel Xeon E3-1220 CPU, 16GByte memory and 500GByte hard-disk. Each slave has 3.2GHz Intel Core i5 CPU, 4GByte memory and 1TByte disk. All machines are connected through a 1Gbps Ethernet switch. Every machine is running on Linux (Ubuntu 10.04 Lucid). We used Hadoop 2.7.3 for the MapReduce framework implementation obtained from [9]. All implemented algorithms were compiled by Javac1.8.

To evaluate the proposed algorithm, we used synthetic data sets. The domain of each dimension is [1, 500,000]. To distribute the points in the domain space, we initially produced 500 $d$-dimensional points randomly. Then, for each initially generated point $p$, we scattered the points around $p$ following the normal distribution, $N(p, 50{,}000^2)$.

**Table 1:** Parameters

| Parameter | Range | Default value |
|---|---|---|
| # of points ($n$) | $1\times10^7$ ~ $5\times10^7$ | $3\times10^7$ |
| # of clusters ($k$) | 10, 20, 30, 40, 50 | 20 |
| # of dimensions ($d$) | 5, 10, 15, 20, 25 | 10 |
| # of iterations ($t$) | 10, 20, 30, 40, 50 | 20 |

To generate d-dimensional data sets as synthetic data sets, we used some parameters, as summarized in Table 1, to make diverse environments. As reported in Table 1, we varied the number of points $n$ from $1\times10^7$ to $5\times10^7$ to show the scalability of our algorithm. In addition, we varied the number of clusters $k$ from 10 to 50 in order to measure the effect of the cluster number. We also varied the number of dimensions d and the number of iterations *t*. The default values of *n*, *k*, *d* and *t* are $3\times10^7$, 20, 10 and 20, respectively.

## 4.2 Experimental Result

In our experiment, we evaluated the execution time of each implemented algorithm. We ran each algorithm five times and report the average execution time of each algorithm in this section.

Varying the number of points (*n*): We varied the number of points n from $1\times10^7$ to $5\times10^7$ and plot the running time of each algorithm in Figure 7. As shown in Figure 7, the execution times of both algorithms increase gradually with increasing n since we have to find the closest center for every point in both algorithms. However, our proposed MPKMeans algorithm is about 11% faster than the previous parallel K-Mean algorithm PKMeans since we alleviate the computational overhead by exploiting the minimum distance from each center to the other centers based on Lemma 1.

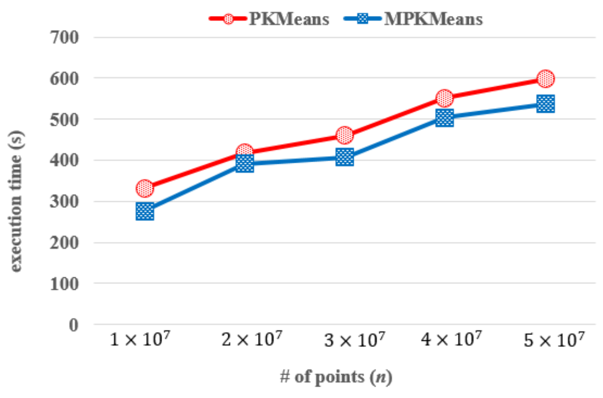

**Figure 7:** Varying n

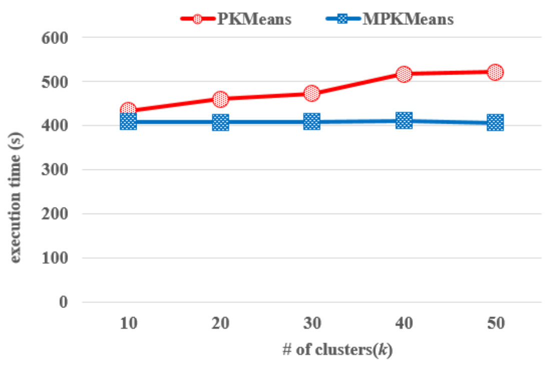

**Figure 8:** Varying *k*

Varying the number of clusters (*k*): Figure 8 shows the experimental result varying the number of cluster *k*. As illustrated in Figure 8, the execution time of PKMeans is increased with increasing the number of cluster *k* since the number of group and communication overhead increase at the shuffle phase as well as the number of reducers each of which computes the center of each cluster is increased. In contrast to PKMeans, since MPKMeans consists of a map phase only over each MapReduce round, the overheads occurred at the shuffle and reduce phases in PKMeans are alleviated. Thus, the performance of MPKMeans is better than PKMeans and stable [22-24].

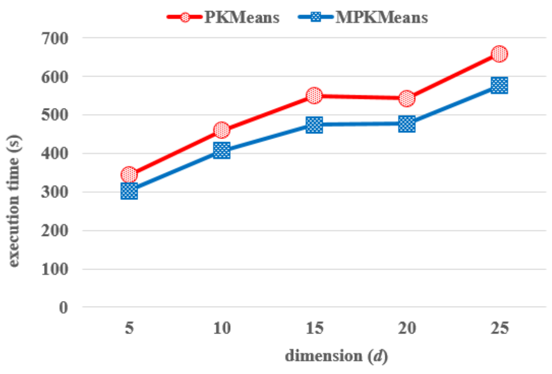

**Figure 9:** Varying d

Varying the number of dimensions (*d*): We also varied the number of dimensions of each point from 5 to 25 with the default values of other parameters. In Figure 9, we plot the execution times of MPKMeans and PKMeans [25].

As shown in Figure 9, the performances of both algorithms are degraded as the number of dimensions increase since the computational overhead for computing distance between each point and ever center in order to find the closest center of each point increases. However, MPKMeans utilizes the closest distance $cmdist_i$ of each center ci to reduce the computational overhead for find the closest center of a point $p$. Only when distance($p$, $c_i$) is greater than $cmdist_i/2$, MPKMeans computes the distance to other centers. Thus, MPKMeans is faster than the previous parallel algorithm PKMeans.

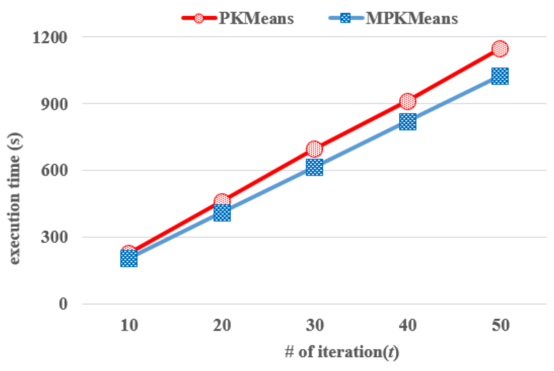

**Figure 10:** Varying t

Varying the number of iterations ($t$): In this experiment, we measure the effect of the number of iterations. As plotted in Figure 10, although the execution times of both algorithms linearly increase as the number of iterations $t$ increases, MPKMeans shows the better performance than PKMeans as well as the performance gap between PKMeans and MPKMeans increases with increasing $t$. Consequently, MPKMeans is superior to PKMeans over all cases.

## 5. CONCLUSION

In this paper, we propose an efficient parallel K-Means algorithm consisting of a single map phase in each MapReduce round. Thus, we can alleviate the overhead for synchronization, data grouping and network transmission during the shuffle and reduce phases. Furthermore, to alleviate the computational overhead, we utilized the distance to the closest center of each center. The calculation of distances between k centers and points in a data set is reduced by exploiting the minimum distance from each center to the other centers since we can guarantee that a center ci is the closest center of a point p if the distance from p to ci is less than the half of the minimum distance from the center ci to the other centers.

## ACKNOWLEDGEMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (NRF-2019R1F1A1062511).

## REFERENCES

1. J. MacQueen. **Some methods for classification and analysis of multivariate observations**, *In Proceedings of Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, no. 14, 1967, pp. 281-297.
2. S. Kant, T. Mahara, V. K. Jain, D. K. Jain, and A. K. Sangaiah. **Leader Rank based k-means clustering initialization method for collaborative filtering**, *Computers & Electrical Engineering*, vol. 69, pp. 598-609, Jul. 2018.

   https://doi.org/10.1016/j.compeleceng.2017.12.001
3. Y. S. Cho, S. C. Moon, S. C. Noh, and K. H. Ryu, **Implementation of Personalized recommendation System using k-means Clustering of Item Category based on RFM**, *In Proceedings of IEEE International Conf. on Management of Innovation & Technology (ICMIT)*, 2012, pp. 373-383.

   https://doi.org/10.1109/ICMIT.2012.6225835
4. L. Mashayekhy, MM. Nejad, D. Grosu, Q. Zhang and W. Shi. **Energy-aware scheduling of Mapreduce jobs for big data applications**, *IEEE Transactions on Parallel & Distributed Systems*, vol. 26, no. 10, pp. 2720–2733, Oct. 2015

   https://doi.org/10.1109/TPDS.2014.2358556

5. D. Van Hieu and P. Meesad. **Fast k-means clustering for very large datasets based on mapreduce combined with a new cutting method**, *In Knowledge and Systems Engineering. Springer International Publishing*, 2015, pp. 287–298.

   https://doi.org/10.1007/978-3-319-11680-8_23

6. M. Tartara and S. Crespi Reghizzi. **Parallel iterative compilation: using MapReduce to speedup machine learning in compilers**, *In Proceedings of Third International Workshop on Mapreduce and its Applications date. ACM: Delft*, the Netherlands, 2012, pp. 33–40. https://doi.org/10.1145/2287016.2287023

7. H. Yao, Q. Duan, D. Li and J. Wang. **An improved k-means clustering algorithm for fish image segmentation**, *Mathematical and Computer Modelling*, vol. 58, no. 3, pp.790–798, Aug. 2013.

   https://doi.org/10.1016/j.mcm.2012.12.025

8. J. Dean and S. Ghemawat. **Mapreduce: Simplifed data processing on large clusters**, *Communication of the ACM*, vol. 51, no. 1, 2008, pp. 107-113.

   https://doi.org/10.1145/1327452.1327492

9. Apache, Apache hadoop, http://hadoop.apache.org, 2010.

10. W. Zhao, H. Ma, and Q. He. **Parallel k-means clustering based on mapreduce**, *In Proceedings of IEEE International Conf. on Cloud Computing*, 2009, pp. 674-679. https://doi.org/10.1007/978-3-642-10665-1_71

11. S. S. Bandyopadhyay, A. K. Halder. P. Chatterjee, M. Nasipuri, and S. Basu, **HdK-means: Hadoop based parallel K-means clustering for big data**, *In Proceedings of IEEE Calcutta Conference (CALCON)*, 2019, pp. 452-456.

    https://doi.org/10.1109/CALCON.2017.8280774

12. Z. Tang, K. Liu, J. Xiao, L. Yang, and Z. Xiao. **A parallel k-means clustering algorithm based on redundance elimination and extreme points optimization employing MapReduce**, *Concurrency and Computation: Practice and Experience*, vol. 29, no. 20, pp. e4109, Mar. 2017.

    https://doi.org/10.1002/cpe.4109

13. S. Lloyd. **Least squares quantization in PCM**, *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129-137, Mar. 1982.

14. C. Elkan. **Using the triangle inequality to accelerate k-means**, *in Proceedings of International Conference on Machine Learning*, 2003, pp. 147-153.

15. G. Hamerly, and C. Elkan. **Learning the k in K-means**, *in Proceedings of Advances in neural information processing systems*, 2004, pp. 281-288.

16. D. Pelleg, and A. W. Moore. **X-means: Extending K-means with Efficient Estimation of the Number of Clusters**, *in Proceedings of the International Conference on Machine Learning*, 200. pp. 727-734.

17. R. E. Kass, and L.Wasserman. **A reference Bayesian test for nested hypotheses and its relationship to the Schwarz criterion**, *Journal of the american statistical association*, vol. 90, no. 431, pp. 928-934, Sep. 1995.

18. D. Arthur and S. Vassilvitskii. **k-means++: The advantages of careful seeding**, *in Proceedings of CM-SIAM symposium on Discrete algorithms*, 2007, pp. 127-1037.

    https://doi.org/10.1145/1283383.1283494

19. S. Shahrivari and S. Jalili. **Single-pass and linear-time k-means clustering based on MapReduce**, *Information Systems*, vol. 60, no. C, pp. 1-12, Aug.-Sep. 2016.

    https://doi.org/10.1016/j.is.2016.02.007

20. P. S. Bradley, and U. M. Fayyad. **Refining Initial Points for K-Means Clustering**, *in Proceedings of International Conf. on Machine Learning*, pp. 91-99, 1998.

21. K. Alsabti, S. Ranka, and V. Singh. **An efficient k-means clustering algorithm**, *Electrical*

*Engineering and Computer Science*. 43, Syracuse University, 1997.

https://doi.org/10.1109/TPAMI.2002.1017616

22. Kormishkina, L. A., Kormishkin, E. D., Gorin, V. A., Koloskov, D. A., Koroleva, L. P. 2019. Environmental investment: the most adequate neo-industrial response to the growth dilemma of the economy. Entrepreneurship and Sustainability Issues, 7(2), 929-948. http://doi.org/10.9770/jesi.2019.7.2(10)

23. Bernardi, A. 2019. The capability approach and organizational climate as tools to study occupational health and safety, Insights into Regional Development 1(2): 155-169. https://doi.org/10.9770/ird.2019.1.2(6)

24. Prakash, G., Darbandi, M., Gafar, N., Jabarullah, N.H., & Jalali, M.R. (2019) A New Design of 2-Bit Universal Shift Register Using Rotated Majority Gate Based on Quantum-Dot Cellular Automata Technology, International Journal of Theoretical Physics, https://doi.org/10.1007/s10773-019-04181-w.

25. Hussain, H.I., Kamarudin, F., Thaker, H.M.T. & Salem, M.A. (2019) Artificial Neural Network to Model Managerial Timing Decision: Non-Linear Evidence of Deviation from Target Leverage, International Journal of Computational Intelligence Systems, 12 (2), 1282-1294.