k-means Performance Improvements with Centroid Calculation Heuristics both for Serial and Parallel environments

Jeyhun Karimov
Computer Engineering Dept.
TOBB University of
Economics and Technology
Ankara,Turkey
Email: jkarimov@etu.edu.tr

Murat Ozbayoglu
Computer Engineering Dept.
TOBB University of
Economics and Technology
Ankara,Turkey
Email: mozbayoglu@etu.edu.tr

Erdogan Dogdu
Computer Engineering Dept.
TOBB University of
Economics and Technology
Ankara,Turkey
Email: edogdu@etu.edu.tr

***Abstract*—k-means is the most widely used clustering algorithm due to its fairly straightforward implementations in various problems. Meanwhile, when the number of clusters increase, the number of iterations also tend to slightly increase. However there are still opportunities for improvement as some studies in the literature indicate. In this study, improved implementations of k-means algorithm with a centroid calculation heuristics which results in a performance improvement over traditional k-means are proposed. Two different versions of the algorithm for various data sizes are configured, one for small and the other one for big data implementations. Both the serial and MapReduce parallel implementations of the proposed algorithm are tested and analyzed using 2 different data sets with various number of clusters. The results show that big data implementation model outperforms the other compared methods after a certain threshold level and small data implementation performs better with increasing k value.**

***Keywords*-k-means, Big Data, Hadoop, MapReduce, Clustering, parallel algorithms, data mining, unsupervised learning;**

I. Introduction

Clustering is the process of separating different objects and grouping similar ones without explicitly stating how they are distributed among classes. When the class boundaries are known, it becomes a classification problem. For that reason, clustering is often referred as unsupervised learning. Even though there are several clustering algorithms exist in the literature, k-means[1] is by far the most famous and widely used one, due to its simplicity and acceptance on several different types of clustering problems.

Our planned achievements in this study are two-fold. First and foremost, we aimed to improve the k-means performance by introducing a centroid calculation heuristic that finds the same clusters faster than the original k-means algorithm. At the same time, we proposed their parallel versions of this improved k-means algorithm that would provide faster performance for a wide range of data sizes and different number of clusters.

The novelty we provide in this paper is related with performance. That is, the numerical results of our models are the same with Standard k-means, but running time is reduced considerably. We proposed two separate algorithms based on the data size. We also implemented both algorithms in distributed environment, to show that they can be parallelized.

The structure of this paper is as follows: After this brief introduction, the literature review on different versions of k-means algorithm and their accomplishments are covered in Section II. In section III, the proposed models are introduced. In Section IV, the proposed algorithms are analyzed and discussed. Section V explains the data sets that are used in this study. Section VI is the Results and Discussions section where the implementation results and analysis with the data sets are provided. Finally, we have the conclusions, suggestions and acknowledgment.

II. Literature Review

k-means [1] is the most widely used algorithm to cluster data. Its simplicity and applicability makes it popular among other algorithms. There are some studies implemented on optimizing different objectives of k-means algorithm such as Euclidean k-medians [2], [3] and geometric k-center [4]. In Euclidean k-medians, the goal is to minimize the sum of distances to the nearest center, and in geometric k-center version, the goal is to minimize the maximum distance from every point to its nearest center. Another research was done to seek a better objective function of k-means [5]. In that particular study, the authors stated that it was not practical to require the solution of clustering to have minimum sum of distance squares of all particles from their centroids. This can be practical when M, N values are small and $k = 2$, where M is the number of data points, N is the number of dimensions and k is the number of clusters. In this paper, authors suggest that seeking local optima such that there is no exchange of a point between clusters, will reduce the sum of squares within a particular cluster.

Even though these different versions of k-means might have advantages over the original k-means, it is realized that considerable performance improvement can be achieved by

parallelizing the process. As a result, several single machine parallel versions of k-means were proposed [6], [7].

Authors use the concept of canopy to divide data into clusters in a computationally cheaper way [8]. After that, clustering is performed on the points that overlap on same canopies. It is shown that, the complexity of the standard k-means algorithm is reduced by $\frac{f^2}{c}$, where c is the number of canopies and f is the average canopy number that each data point falls to. In general c is much larger than f. Authors show in the experiment that was done on $n = 1,000,000$, $k = 10,000$, $c = 1,000$, and f is a small constant, the canopies technique reduces the amount of computation by a factor of $1,000$, where n is the number of data points, k is the number of clusters

Another improvement to k-means algorithm is the kd-tree implementation using the filtering algorithm [9]. It is based on storing the data points on the kd-tree [10]. It hierarchically divides the point set using axis aligned splitting hyperplanes. So, for n points, the algorithm produces a tree with $O(n)$ nodes and $O(log(n))$ depth. Authors show that experimental results were significantly better than other traditional approaches to clustering.

Yet another improvement to k-means algorithm is proposed by concentrating on the algorithm's shortcomings [11]. Authors of this research stated that despite the algorithm's popularity, it suffered from several issues. According to the authors, these problems were its computationally inadequate scalability, dependency on the number of clusters K and its prone to local minima. Their proposed model - X-Means- provided the number of classes and their parameters in a fast and statistically established way. X-means consists of three main parts:

1) Improve-Params
2) Improve-Structure
3) If K is less than the maximum number of K, stop and report the best scoring model found during the search where Improve-Params runs conventional k-means to convergence, Improve-Structure finds out where new centroids should appear by splitting some centroids.

Even though there were several attempts to improve the performance of k-means, since k-means (and its improved versions) can be parallelized easily and very effectively due to its nature, it is possible to have more significant performance improvement achievements through parallelization of the serial versions of the algorithm. This can be achieved with an framework like MapReduce [12], [13].

There has been several studies for clustering large scale data on distributed systems in parallel on Hadoop [14]. One such approach is Haloop [15], which is a modified version of the Hadoop MapReduce framework. The proposed model dramatically improves the efficiency by making the task scheduler loop-aware and by adding various caching mechanisms. Authors used the k-means algorithm to evaluate their model against the traditional one and as a result, the proposed model reduced the query runtimes by 1.85.

Another approach to cluster data in a distributed system was using Apache Mahout library. Research was done to cluster the data in the cloud [16]. The tests were running on Amazon EC2 instances and the comparisons were made to realize the gain between the nodes. Yet another study was done to cluster Wikipedia's latest articles with k-means [17].

Another research was concentrated on MapReduce model's lack of directly supporting processing multiple related heterogeneous datasets [18]. Authors called their model Map-Reduce-Merge. It adds a Merge phase to the standard model. This phase can efficiently merge the data already partitioned and sorted by the map and reduce modules.

There are advantages of MapReduce over parallel databases like storage-system independence and fine-grain fault tolerance for large jobs [19]. Since MapReduce model works on multicore systems, some researchers evaluated the suitability of the this model for multi-core and multi-processor systems [20]. Authors of this research studied Phoenix with multi-core and symmetric multiprocessor systems. Afterwards, they evaluated its performance potential and error recovery features. Moreover, they also compared the codes of MapReduce and P-threads which was written in lower-level API. As a result, authors concluded that MapReduce was a promising model for scalable performance on shared-memory systems with simple parallel code. MapReduce model is mostly used in offline jobs, due to its efficient processing of large data and late response time. However, authors of [21] researched the online version of Hadoop MapReduce framework.They propose a solution that allows users to see 'early returns' from a job while it is being computed and process continuous queries on the framework. Authors of [22] also used MapReduce framework to implement k-means in parallel which is one of the models we used to evaluate and compare our solution.

III. PROPOSED MODEL

Although there are numerous modifications of the k-means algorithm, both in single machine and in MapReduce model, the complexity of the algorithm more or less remains the same. In particular, Standard k-means (k-means-s) model can not escape from the complexity of the algorithm where the new centroids and nearest centroids are recalculated each time [22]. So, in each iteration:

- First part (P1) - All points are processed and the nearest centroids are found
- Second part (P2) - All points are processed in k groups to find the new centroids

As can be seen above, reprocessing of all points twice in each iteration increases the computation time of the algorithm linearly as the data and k value gets bigger. Improvement for the first part was done in [23]. However, this improvement have some disadvantages when working

with big data. That is why, we further improved the model proposed in [23] and proposed two new solutions for the parallel computation with big data and the serial computation with non-big data. We will show the threshold between big data and non-big data for the data sets we used, in Section VI.

The models we propose, namely k-means-inbd (k-means improved for non-big data or k-means Improved for Serial Computation) and k-means-ibd (k-means improved for big data or k-means Improved for Parallel Computation), eliminates the majority of the complexity associated with both parts of the standard parallel k-means [22]. This improvement decreases the computation time and the complexity of the algorithm considerably. So, let

- x_i denote a single data point and S denote all set of points in data set,where $x_i \in S \forall i$.
- $c_j^t \in C^t$ denote the centroid computed in t^{th} iteration, where $j = \{1, 2, 3, ..., k\}$
- S_j^t denote the set of data points belonging to c_j^t
- P_j^t denote the set of newly accepted points to cluster representing c_j^t
- M_j^t denote the set of outgoing points from cluster representing c_j^t
- z_i denote the distance between x_i and its belonging centroid.
- v_i denote the index of x_i^{th} belonging centroid, c_k^t from the set C^t
- α be a constant value denoting threshold value where $0 < \alpha < 1$.

The first proposed model is k-means-inbd. General procedure of this model is as follows:
In the first iteration of k-means-inbd, for all data points, their nearest centroids are calculated and for each $x_i \in S$, we keep z_i and v_i. After that, new centroids are calculated as in k-means-s. Beginning from $t = 2^{th}$ iteration, when computing the nearest centroids for each data point, $x_i \in S$, we calculate $d^t(x_i, c_{v_i}^t)$, distance between current data point and its previous centroid's new value. That is, $c_{v_i}^t$ is newly computed value of centroid $c_{v_i}^{t-1}$. If $d^t \leq z_i$, then x_i stays in same cluster. Thus, it can be ignored during the recalculation of the new centroid. Otherwise, it means that x_i has changed its cluster and it must be considered while recalculating the new centroids. After processing all data points, only those that were chosen are considered in the calculation of the new centroids. So, the calculation of a new centroid is shown with Formula (1):

$$c_j^t = \frac{c_j^{t-1} * |S_j^{t-1}| - (\sum_{i=1}^{a} m_i^t) + (\sum_{i=1}^{b} p_i^t)}{|S_j^{t-1}| - a + b} \quad (1)$$

where $c_j^t \in C^t$ is the j^{th} centroid among k centroids at t^{th} iteration, $|S_j^{t-1}|$ is the number of points belonging to c_t^{t-1}, $m_i^t \in M_j^t$ is i^{th} point that is drawn out from j^{th} cluster at t^{th} iteration, $p_i^t \in P_j^t$ is i^{th} point that is added to j^{th} cluster at t^{th} iteration, $b = \left|P_j^t\right|$ and $a = \left|M_j^t\right|$. The pseudocode of k-means-inbd is shown in Algorithm (1).

Algorithm 1

procedure k-MEANS-INBD($x_i \in S, \forall i$, k)
  Require: $S = \{x_0, x_1 ... x_n\}$, k is number of clusters
  Ensure: $c_1, c_2, ..., c_k$ centroids
  Initialize centroids for $t = 1$.
  Run k-means with its standard execution for the first iteration and keep z_i and v_i $\forall$ $x_i \in S$.
  Initialize C^t, set of resulting centroids at the end of iteration $t = 1$.
  while $c_j^t \neq c_j^{t-1}$ $\forall j$ **do**
    $t = t + 1$.
    for all $x_i \in S$ **do**
      Compute distance $d^t = d(x_i, c_{v_i}^t)$ and $d^{t-1} = z_i$.
      if $d^t \leq d^{t-1}$ **then** continue.
      else
        Compute $c_b^t \in C^t$, x_i^{th} new associated centroid from set C^t, where $b \neq j$.
        $P_b^t = P_b^t \bigcup x_i$, add x_i to set of new coming points for centroid c_b^t
        $M_j^t = M_j^t \bigcup x_i$, add x_i to the set of outgoing points for centroid c_j^{t-1}.
      Save x_i^{th} associated z_i and v_i, to use in the next iteration.
    Compute the new centroids using Formula (1).

The second proposed model is k-means-ibd (k-means improved for big data). The general structure of algorithm is as follows:

- Before threshold part (BT) - For i iterations until reaching threshold value, run as k-means-inbd. At each iteration t, compute threshold value α_t.
- After threshold part (AT) - if $\alpha_t > \alpha$, then run Algorithm (3).

The first i iterations until reaching threshold value is as the same as k-means-inbd. Beginning from $(i+1)^{th}$ iteration, we store centroids of previous iteration and size of all clusters. The iteration number i is decided as a result of threshold value α. That is, if the division of data points that changed their clusters to all data points in t^{th} iteration ($\alpha_t = \frac{|P^t|}{|S|}$) is less than predefined threshold (α), then we can be confident about clusters' being mostly stable. Important point is that, after threshold value is satisfied, in AT part, we do not keep all points' associating centroids, but the set of newly computed centroids, which can fit in memory in big data sets. In AT part, when recomputing the object assignments to the new centroids, the first to consider is the previous centroid. First we compute x_i^{th} previous nearest centroid, c_j^{t-1}. When computing the new centroid, begin from j^{th}

Algorithm 2

procedure k-MEANS-IBD($x_i \in S, \forall i$, k)
Require: $S = \{x_0, x_1 ... x_n\}$, k is number of clusters
Ensure: $c_1, c_2, ..., c_k$ centroids
while $\alpha_t > \alpha$ **do**
Run Algorithm (1) for 1 iteration.
Compute $\alpha_t = \frac{|P^t|}{|S|}$, overall fraction of points that changed their existing clusters. Here $|P^t| = |M^t|$
Run Algorithm (3) with required parameters from this algotihm.

centroid from C^t, namely c_j^t. If $d^{t-1}(x_i, c_j^{t-1}) \geq d^t(x_i, c_j^t)$, it means that the x_i stayed in the same cluster and there is no need to consider this data point when computing the new centroids. Otherwise, the data point has changed its cluster and it must be considered while recalculating a new centroid. Recalculating the new centroid part is the same as k-means-inbd. The pseudocode of k-means-ibd is shown in Algorithm (2).

Since we compared both the parallel and the serial versions of the proposed models, MapReduce version of k-means-ibd and k-means-inbd was also implemented. The algorithm is the same. However, the mapping of the serial version's first part (evaluating each data point's nearest center) to the parallel version's mapper phase and the serial version's second part (calculating the new centroids after all data points chose their nearest centers) to the parallel version's reducer phase is enhanced. That is, in the mapper phase we find the data point's nearest centroid and in the reducer phase the new centroids are calculated from the points that changed their cluster.

IV. ANALYSIS OF PROPOSED MODELS

When analyzing the proposed models, we can divide the k-means-inbd and k-means-ibd into two parts as k-means-s:

- First part - All points are processed and the nearest centroids are found.
- Second part - All points are processed in k groups to find the new centroids.

Both k-means-inbd and k-means-ibd have improvements in the second part. If we examine Figure 1 which explains Formula (1), it can be seen that there are two main subparts denoted as 1 and 2. First, denoted as 1, is a computationally constant time operation. As we will discuss in the experimental results in this paper, it is seen that second part, denoted as 2, consists of a small minority of the points of the whole data set, so as the iterations progress, the number of operations keep decreasing geometrically, i.e. the total number of operations converges to a constant; hence the whole formula can become a constant time operation.

Algorithm 3

procedure k-MEANS-IBD-AT($x_i \in S, \forall i$, k, C^t)
Require: $S = \{x_0, x_1 ... x_n\}$, k is number of clusters, C^t set of current centroids.
Ensure: $c_1, c_2, ..., c_k$ centroids
Initialize centroids for $t = 1$.
while $c_j^t \neq c_j^{t-1}$ $\forall j$ where $0 < j \leq k$ **do**
$t = t + 1$.
for all $x_i \in S$ **do**
Among previous centroids, find the nearest centroid $c_j^{t-1} \in C^{t-1}$ to x_i and compute distance between them $d_j^{t-1} = d(x_i, c_j^{t-1})$.
Find $d_j^t = d(x_i, c_j^t)$, distance from x_i to j^{th} cluster from C^t, where j is index of c_j^{t-1}.
if $d_j^t \leq d_j^{t-1}$ **then** continue.
else
Compute $c_b^t \in C^t$, x_i^{th} new associated centroid from set C^t, where $b \neq j$.
$P_b^t = P_b^t \bigcup x_i$, add x_i to set of new coming points that belong to centroid c_b^t.
$M_j^t = M_j^t \bigcup x_i$, add x_i o the set of outgoing points that belonged to centroid c_j^{t-1}.
Save only the set C^t to be used in the next iteration.
Compute the new centroids using Formula (1).

$$c_j^t = \frac{c_j^{t-1} * |S_j^{t-1}| - (\sum_{i=1}^{a} m_i^t) + (\sum_{i=1}^{b} p_i^t)}{|S_j^{t-1}| - a + b}$$

Figure 1. Improvement done on second part of k-means algorithm

While analyzing the first part of proposed models, in k-means-ibd model, we are interested in minimizing the overall data sent from $mapper$ to $reducer$ phase and minimizing I/O time. The main advantage of k-means-ibd is that, it does not change the original data after threshold is satisfied. So, there is no disc-write overhead, in any iteration after $\alpha_t < \alpha$. Important point here is that, after threshold value is satisfied, overall points tent to stay in their existing clusters, as we will see in Section VI. Therefore, we switch to Algorithm (3). This algorithm keeps only the previous centroids set which can be kept in the memory for very large data sets. That is why, as the size of the data gets bigger, k-means-ibd starts outperforming k-means-inbd in MapReduce parallel computing model. k-means-inbd on the other hand, have less complexity compared to k-means-ibd. Because k-means-

inbd model keeps all data points' previous centroids, after several steps, calculation of the new centroids is taking $O(1)$ instead of $O(k)$ time most of the time. However k-means-inbd has an obvious space disadvantage. That is, when working with big data, in every iteration, all data points' centroids must be read and written to the disc, since they can not be kept in the memory. As it will be seen in Section VI, there is threshold value for data set depending on the overhead of writing big data to disc that dominates over k-means-inbd's improvement in the first part of the algorithm. That is why, we considered this algorithm to be the best for the serial implementation and for the parallel implementation with upper bound data size.

As demonstrated in [24], the worst-case running time of k-means is superpolynomial by improving the best known lower bound from $\Omega(n)$ iterations to $2^{\Omega(\sqrt{n})}$. That is, k-means always has an upper bound, therefore it always converges. So, because it always converges, the displacement speed of centroids must go to zero as iterations go to some finite number. Therefore, their speed must decrease, otherwise the algorithm cannot converge. Because centroids' speed decrease, the points that belong to particular centroid, tend to stay in that cluster.

V. SYSTEM SETUP AND DATASETS

The experiments were conducted both in serial and parallel environment. MapReduce framework of Cloudera's Apache Hadoop distribution was used for parallel environment. The environment consisted of 17 connected computers with $100Mbit/s$ Ethernet. Each computer had Intel i7 CPU and 4GB RAM capacity. Among the 17 computers, 16 of them were worker nodes and 1 was the master node.

Two different data sets were used to run the experiments. First data set (DS-1) was, "Individual household electric power consumption Data Set" [1] and the second one (DS-2) was, "US Census Data (1990) Data Set" [2]. The lengths of feature vectors of DS-1 and DS-2 are 7 and 68 and the size of data sets are 2075259 and 2458285 instances respectively. Both data sets were divided into different number of clusters and the algorithms run with different initial centroids. Finally, we chose α threshold to be 0.15 in experiments.

VI. RESULTS AND DISCUSSION

We have performed numerous experiments both in serial and parallel environments. We compared our proposed improvements with the models proposed in [23], [25] and standard k-means - (k-means-s) [22]. The complexity and efficiency of the models described in [23] and [25] are mainly the same. Therefore we implemented the model shown in [23] to compare with our proposed algorithms. As the authors of [23] called their model as enhanced k-means, for simplicity we called their model as k-means-e.

Before discussing the results, one important thing is that, there is no k-means-s in figures, because the graphs show relative results with respect to k-means-s. Since in all of the fore-mentioned models we are trying to achieve improvements over k-means-s, all graphs shown in this section used k-means-s performance as the basis. This is accomplished by dividing the result of the particular model running time by the running time of k-means-s. This can also be considered as a normalization.

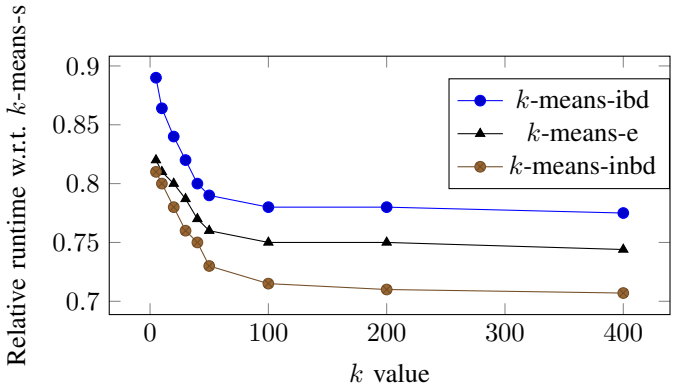

Figure 2. Comparison of three models in serial environment with DS-1.

Figure 2 shows the comparison of our proposed models and k-means-e [23] in terms of their efficiency towards k-means-s [22] with DS-1 in the serial environment. It is clear that k-means-ibd is less efficient compared to the other two models. As stated above, all proposed models consist of two parts: first part and second part. Here k-means-ibd mainly takes advantage of the improvement of the second part when compared to k-means-s. k-means-e also takes advantage of the improvement of the first part, when finding the nearest centroids. However, k-means-inbd is improved both in the first and the second part, that is why, it performs better than the other models. In general, it is obvious that when the data is small compared to the memory size and in the serial environment, first part of the models dominates the second part. That is why, even though k-means-ibd performs better than k-means-s, it is still slower than the other two models.

Figure 3 shows the same procedure as Figure 2 with DS-2. It is clearly seen that the overall concept is pretty much the same. Meanwhile, all models more or less have improved their performance slightly by decreasing their computation time. This is due to the fact that the feature vector size in DS-2 was 68, whereas it was 7 in DS-1. However, k-means-inbd and k-means-e improved their computation time more than k-means-ibd, when compared to Figure 2 because k-means-e and k-means-inbd benefit computationally over k-means-s in part-1 which is directly related to the vector size, more than k-means-ibd. As the vector dimension increases,

[1] https://archive.ics.uci.edu/ml/machine-learning-databases/00235/

[2] https://archive.ics.uci.edu/ml/machine-learning-databases/census1990-mld/

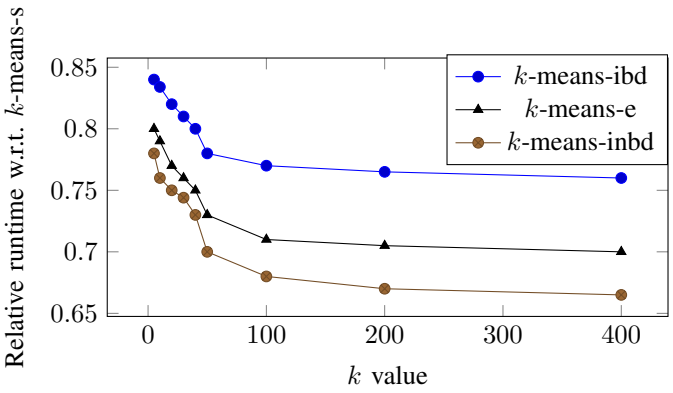

Figure 3. Comparison of three models in serial environment with DS-2.

k-means-inbd and k-means-e have more dominance over k-means-s, since the first part of the proposed models have dominance over the second part in the serial environment.

The performance improvement over standard k-means increases with larger k values due to the increase in the number of iterations to converge, as seen in Figures 2 and 3. As the number of iterations increase, we have much more benefit using our proposed models compared to standard k-means.

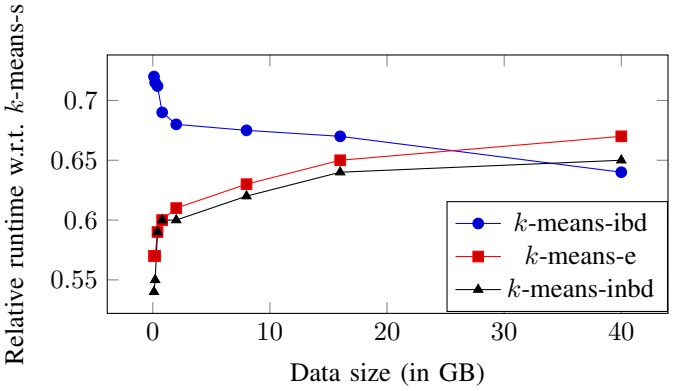

Figure 4. Comparison of three models in parallel environment with DS-1.

Figure 4 shows the comparison of the proposed models over k-means-s in the parallel environment. We used Cloudera's Hadoop distribution with 17 nodes in this experiment. The main purpose of this experiment was to find the threshold value for the size of the data set where k-means-ibd starts outperforming k-means-inbd. Therefore we simulated DS-1 to have a larger data set. As stated during the analysis of the algorithms, the main disadvantage of k-means-e and k-means-inbd is their necessity to keep all data points' previous centroids. If the serial environment is used with a data size that is less than the memory size, they can be kept in the memory. However with the increasing data size, it will not be possible to achieve that. In MapReduce implementation, k-means-e and k-means-inbd change the data set with their modified centroids and in every iteration output the result. That is, in every iteration, stated models read $O(2n)$ data points and output $O(2n)$ data file size. As the size of the data file gets larger, the dominance of the improvements keep decreasing due to the increasing I/O time to output and write to the large disc files. Moreover, reading time is also increased due to extra $O(n)$ data read in each iteration. As the iteration number gets larger to converge, this drawback becomes a major issue for k-means-s and k-means-inbd. We can see that in Figure 4 for DS-1, k-means-ibd keeps getting slightly better as the data size increases, because of the improvement in I/O and in the $reducer$ side of MapReduce. However, k-means-e and k-means-inbd have a deteriorating performance with the increase in the data size due to the reason stated above. Another notable observation is, k-means-e's data size threshold being less than k-means-inbd. This is mainly due to the fact that it has no improvements in the reducer side and all points are sent from the mapper to the reducer in every iteration. However, in k-means-inbd as well as in k-means-ibd only those that have changed their cluster are considered for processing in the reducer. The reducer and the partition phase take longer when the data set becomes larger. However, as k-means-ibd does not send all data points from the $mapper$ to the $reducer$, it takes advantage of the reducer phase improvement and this advantage becomes more significant with increasing data size. Also, k-means-ibd has an I/O advantage over the other models, because of getting rid of reading $O(n)$ and writing extra $O(2n)$ data in each iteration.

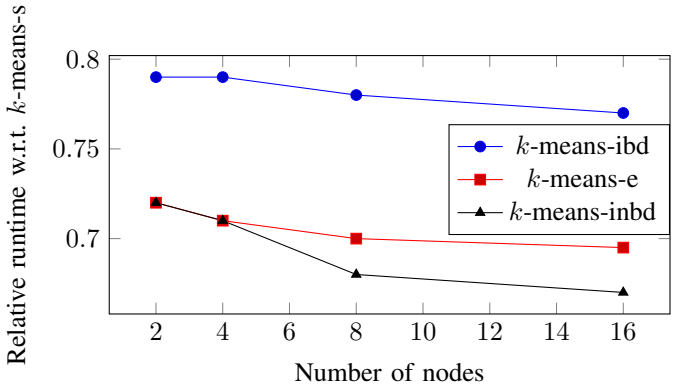

Figure 5. Comparison of three models in parallel environment with different number of nodes with k=100 and data DS-2.

Figure 5 shows the comparison of the proposed models' computation time over k-means-s with different number of nodes using MapReduce. In Figure 5, we used DS-2 with its size simulated up to $400MB$ and $k = 100$ in this graph. We simulated the data set in order to see real outputs that were less influenced by the network overhead. Here it is seen that k-means-inbd and k-means-ibd have increasing performance

improvement over k-means-s as the number of nodes in the cluster increase. The main reason for this is, if we have $O(x)$ improvement in one node and if we distribute the job to m nodes, we will have $O(x*m)$ improvement, not considering the network overheads. However, the improvement of k-means-e is less than k-means-inbd as the number of nodes increase. Again the reason is k-means-e's not getting the advantage of the reducer side improvement. Overall picture is the same for data set 1, but again, have slight improvements in k-means-ibd and more improvements in k-means-e and k-means-inbd. The reason can be explained as follows: k-means-inbd and k-means-e have mapper improvements, that is why, they outperform k-means-ibd in small data sizes (in this particular case, with a data size of $400MB$). As we increase the size of data set, the relative performance of k-means-ibd increases and after the size threshold value, it outperforms other algorithms.

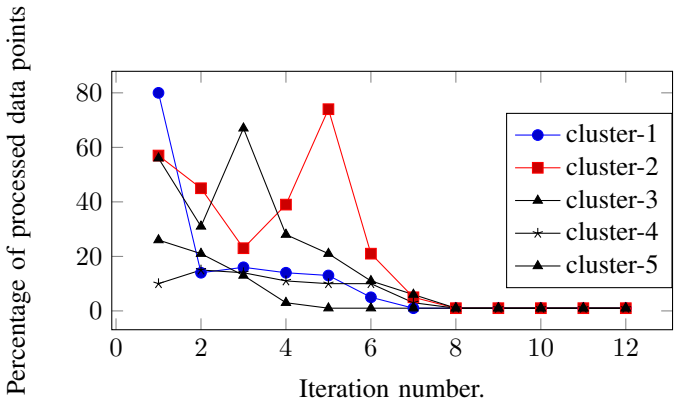

Figure 6. Percentage of points changed in clusters in reduce step for k=5.

Figure 6 shows the percentage of data points that was processed in the $reduce$ step in both k-means-ibd and k-means-inbd for $k = 5$. This case was chosen as an example to demonstrate the general concept in a realistic environment. In that particular graph, it can be observed that after a certain iteration, all data sets had decreasing number of operations performed at each following iteration. This was our main motivation to the improvement achieved on the second part of the proposed models, namely, processing of only those points which altered their clusters to compute the new centroids. Since not all of the points are considered in finding the new centroid (which is not the case in standard k-means), after several iterations the number of points that has changed their cluster decreases drastically. If this graph had included k-means-s, all lines would be ($percentage = 100$) straight lines; that is, no matter which iteration was carried, all $reducers$ process all points. However, in our proposed models, as the iteration numbers increase, clusters tend to converge and the number of operations reduce geometrically. For simplicity, we show the graph until the 12^{th} iteration, because after that, points in all the clusters change less than 1 percent until convergence. It is obvious from the graph that, as the number of iterations increase, our proposed models demonstrate better efficiency.

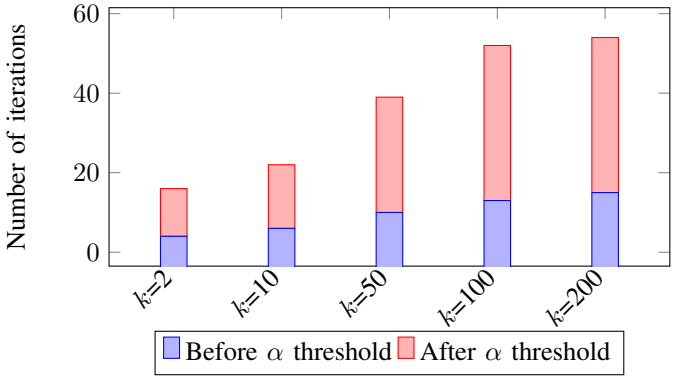

Figure 7. Number of iterations before and after $\alpha = 0.15$ threshold with DS-2.

Figure 7 shows the number of iterations needed to satisfy threshold α and the ones needed to converge. This is the point where k-means-ibd switches from Algorithm (1) to Algorithm (3). Since k-means algorithm is greedy, it converges at the first local minimum. Therefore after a few steps, the clusters tend to be stable as their centroids tend to move more slowly. So, with increasing k values, we have more iterations after threshold. This means that we can take more advantage on $reducer$ side at each step.

VII. CONCLUSIONS

In this study, an improvement over the standard k-means clustering algorithm is suggested. In particular, instead of using the full data set in the centroid updating step of the algorithm, only the data points which will change their cluster (by associating themselves to a different cluster centroid) would be considered. This adjustment on the algorithm provides a considerable efficiency. We used two different versions of this algorithm depending on the data size, cluster size and serial or parallel environment. We call these algorithms k-means-inbd and k-means-ibd. Furthermore, both serial and parallel implementations were implemented; MapReduce was used for this purpose.

The results indicate that using this new enhanced algorithm not only provided a considerable performance improvement over the classic algorithm but also outperformed an improved k-means algorithm from the literature in all tested cases. When the number of the iterations increased, the amount of work k-means had to perform increases linearly. Also increasing the number of clusters resulted in higher number of iterations, hence, it resulted in a linear increase in workload for k-means. However, since the amount of work performed by the proposed algorithms tend to become smaller at each iteration (after the initial

adjustment period), the total amount of work that needed to be performed compared to k-means kept decreasing.

Similar results were obtained when implementing MapReduce using both datasets. The performance improvement got better and the results were more significant when the dataset size became larger.

Even though the results are experimental and it was tested on two data sets, models showed consistent performance on all different tested situations. Further analysis might be required, but the improvement over the classical and the improved k-means and the advantages of MapReduce implementations are noteworthy.

ACKNOWLEDGMENT

This study is part of the research in the SANTEZ-0367.2013-2 project that is funded by the Turkish Ministry of Science, Industry and Technology.

REFERENCES

[1] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. California, USA, 1967, pp. 281–297.

[2] S. G. Kolliopoulos and S. Rao, "A nearly linear-time approximation scheme for the euclidean k-median problem," in *Algorithms-ESA'99*. Springer, 1999, pp. 378–389.

[3] S. Arora, P. Raghavan, and S. Rao, "Approximation schemes for euclidean k-medians and related problems," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 106–113.

[4] P. K. Agarwal and C. M. Procopiuc, "Exact and approximation algorithms for clustering," *Algorithmica*, vol. 33, no. 2, pp. 201–226, 2002.

[5] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Applied statistics*, pp. 100–108, 1979.

[6] X. Li and Z. Fang, "Parallel clustering algorithms," *Parallel Computing*, vol. 11, no. 3, pp. 275–290, 1989.

[7] C. F. Olson, "Parallel algorithms for hierarchical clustering," *Parallel computing*, vol. 21, no. 8, pp. 1313–1325, 1995.

[8] A. McCallum, K. Nigam, and L. H. Ungar, "Efficient clustering of high-dimensional data sets with application to reference matching," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2000, pp. 169–178.

[9] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 7, pp. 881–892, 2002.

[10] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[11] D. Pelleg, A. W. Moore *et al.*, "X-means: Extending k-means with efficient estimation of the number of clusters." in *ICML*, 2000, pp. 727–734.

[12] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[13] R. Lämmel, "Google's mapreduce programming model—revisited," *Science of computer programming*, vol. 70, no. 1, pp. 1–30, 2008.

[14] T. White, *Hadoop: the definitive guide: the definitive guide*. " O'Reilly Media, Inc.", 2009.

[15] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst, "Haloop: Efficient iterative data processing on large clusters," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 285–296, 2010.

[16] R. M. Esteves, R. Pais, and C. Rong, "K-means clustering in the cloud–a mahout test," in *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*. IEEE, 2011, pp. 514–519.

[17] R. M. Esteves and C. Rong, "Using mahout for clustering wikipedia's latest articles: a comparison between k-means and fuzzy c-means in the cloud," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. IEEE, 2011, pp. 565–569.

[18] H.-c. Yang, A. Dasdan, R.-L. Hsiao, and D. S. Parker, "Map-reduce-merge: simplified relational data processing on large clusters," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 1029–1040.

[19] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[20] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating mapreduce for multi-core and multiprocessor systems," in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*. IEEE, 2007, pp. 13–24.

[21] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online." in *NSDI*, vol. 10, no. 4, 2010, p. 20.

[22] W. Zhao, H. Ma, and Q. He, "Parallel k-means clustering based on mapreduce," in *Cloud Computing*. Springer, 2009, pp. 674–679.

[23] A. Fahim, A. Salem, F. Torkey, and M. Ramadan, "An efficient enhanced k-means clustering algorithm," *Journal of Zhejiang University SCIENCE A*, vol. 7, no. 10, pp. 1626–1633, 2006.

[24] D. Arthur and S. Vassilvitskii, "How slow is the k-means method?" in *Proceedings of the twenty-second annual symposium on Computational geometry*. ACM, 2006, pp. 144–153.

[25] C. Elkan, "Using the triangle inequality to accelerate k-means," in *ICML*, vol. 3, 2003, pp. 147–153.