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

—*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

978-1-4673-7278-7/15 $31.00 © 2015 IEEE
DOI 10.1109/BigDataCongress.2015.72

IEEE computer society

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

—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

978-1-4673-7278-7/15 $31.00 © 2015 IEEE
DOI 10.1109/BigDataCongress.2015.72

IEEE computer society

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

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 $j = \{1, 2, 3, \ldots, k\}$
- 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
- a be a constant value denoting threshold value where $0 < a < 1$.

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 $d^t(x_i, c^t_{v_i})$, distance between current data point and its previous centroid's new value. That is, $c^t_{v_i}$ is newly computed value of centroid $c^{t-1}_{v_i}$. 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, $b = P_j^t$ and $a = M_j^t$. The pseudocode of *k*-means-inbd is shown in Algorithm (1).

Algorithm 1

```
1:  procedure k-MEANS-INBD(x_i ∈ S, ∀i , k)
        Require: S = {x_0, x_1...x_n}, k is number of clusters
        Ensure: c_1, c_2, ..., c_k centroids
2:      Initialize centroids for t = 1.
3:      Run k-means with its standard execution for the first
        iteration and keep z_i and v_i ∀ x_i ∈ S.
4:      Initialize C^t, set of resulting centroids at the end of
        iteration t = 1.
5:      while c_j^t = c_j^{t-1} ∀j do
6:          t = t + 1.
7:          for all x_i ∈ S do
8:              Compute distance d^t = d(x_i , c_i^t) and d^{t-1} = z_i
9:              if d^t ≤ d^{t-1} then continue.
10:             else
11:                 Compute c_b^t ∈ C^t, x_i^th new associated centroid from
                    set C^t, where b = j.
12:                 P_b^t = P_b^t ∪ x_i, add x_i to set of new coming points
                    for centroid c_b^t
13:                 M_j^t = M_j^t ∪ x_i, add x_i to the set of outgoing points
                    for centroid c_j^{t-1}.
14:             Save x_i^th associated z_i and v_i, to use in the next
                iteration.
15:         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 a_t.
- After threshold part (AT) - if $a_t > a$, 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 a. That is, if the division of data points that changed their clusters to all data points in t^{th} iteration ($a_t = \frac{P^t}{|S|}$) is less than predefined threshold (a), 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}

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：https://d.book118.com/688040127002006036