An Efficient Distributed Data Clustering Algorithm

Malay K. Pakhira

Kalyani Government Engineering College, Kalyani, India
Email: malay_pakhira@yahoo.com

Abstract—The k-means algorithm is one of the most popular clustering algorithms in use today. The high running time complexity of serial k-means limits its applicability for very large databases. On the other hand, the existing parallel k-means algorithms demand huge data transfer operations incorporating high communication complexity. Transfer of actual data from local sites is also unacceptable, in many situations, on security and privacy grounds. This work proposes a distributed clustering algorithm that can be executed in a distributed network of processors to achieve significant reduction in computation and communication times while identifying the cluster structure inherent to a data set. The large amount of data transfer, involved in parallel k-means algorithms, is avoided to reduce the communication overhead as well as to ensure better security and privacy. Results of experimentation show that the proposed distributed approach can provide higher speedup than other reported algorithms and can effectively be employed in large applications.

Index Terms—Complexity, Distributed environment, Distributed clustering, Large databases.

I. INTRODUCTION

Clustering [1-4] is an unsupervised classification mechanism where a set of objects (patterns), usually multidimensional in nature, are classified into groups (classes or clusters) such that members of one group are similar according to a predefined criterion. Clustering of a set forms a partition of its elements chosen to minimize some measure of dissimilarity between members of the same cluster. Clustering algorithms are often useful in various applications like data mining, pattern recognition, learning theory etc. The most commonly-used clustering algorithm is the k-means algorithm[5]. It is most effective for relatively smaller data sets.

There exists different parallel versions of the k-means algorithm [6-10], to reduce its time complexity and to make it useful for large applications. In [6] a constant time k-means algorithm is implemented on a reconfigurable array of processors. The algorithm requires at least as many processors as the number of data. Kantabutra and Couch presented a parallel k-means algorithm [7] on NOWs (Network of Workstations) employing K number of slave machines, where K is the number of clusters. It requires broadcasting of local data set over the Ethernet at each iteration and uses global exchange mechanism. Such an algorithm cannot be so effective when the input size increases and is prone to privacy and security attacks. In [8], an improvement in terms of speedup has also been reported with a parallel k-means on COWs (Cluster of Workstations).

The distributed clustering technique presented in [11], prohibits transmission of data between processors in order to maintain security measures. An EM (Expectation Maximization) based clustering is used to find local clusters as Gaussian distributions. After local processing, a centralized merging process determines the merge-able sub-clusters in different processors. However, the EM is time consuming and in order to find the appropriate cover for the Gaussian, EM is executed for a number of times. Further, the asymptotic rate of convergence of EM is very slow (critically dependent on initialization) [12].

Another approach to reduce the iteration complexity of the k-means is found in [13]. Here, at the local processors a sample based clustering is done to ensure near k-means center vectors. The center vectors are then used to run the original k-means on the complete data set to find out global clusters. The process of ensuring correct k-means centers from the sampled data, however, involves complicated processes like measuring confidence radius etc. Several other attempts to improve speedup in clustering are found in [14-17]. The most useful contribution, in this regard, is due to Dhillon and Modha [16]. Here, the k-means is executed in a distributed fashion, using broadcast of limited information among the processors.

We have introduced a variation in center computation strategy of k-means that reduces amount of data transfer and thereby yields a more secured and very fast algorithm. The proposed distributed algorithm can be implemented using any number of processors P, where P ≈ N for practical considerations. The proposed scheme can attain a maximum speedup ≈ the number of processors used in the distributed system. The data set to be clustered is assumed to be distributed among the processors in a network. The local data at each processor are assigned to the nearest representative local center vector. At each iteration, once the local clusters are formed, new centers are computed considering the old ones as data items. Each processor then broadcasts its own center vectors to every other processor and computes a set of average centers parallely.

The organization of this paper is as follows. In Section II, we present a brief introduction to the serial k-means algorithm and describe some of its parallel versions. In Section III, we describe our proposed scheme for data clustering in a distributed system. We report modifications of the basic k-means algorithm necessary for developing the distributed algorithm without violation of any of its desired characteristics. The complexity analysis is given in Section IV, followed by a...
comparative study of the effectiveness as well as limitations of conventional schemes, so far been reported, in Section V as experimental results. Conclusions are drawn in Section VI.

II. THE k-MEANS ALGORITHM FOR CLUSTERING

The k-means algorithm [5] is very commonly used for clustering data. To handle a large data set, a number of different parallel implementations of the k-means have also been developed. In crisp partitional clustering, a set \( \mathbf{D} \) of \( \mathbf{N} \) patterns \( \{ \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \} \) of dimension \( d \) is partitioned into \( K \) clusters denoted by \( \{ C_1, C_2, \ldots, C_K \} \) such that the sum of within cluster dispersions, i.e., the clustering metric \( \mathcal{M} \), as given in (1), becomes the minimum.

\[
\mathcal{M} (C_1, C_2, \ldots, C_k) = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \| \mathbf{x}_i - \mathbf{z}_k \|^2 \quad (1)
\]

Here, \( \mathbf{Z} = \{ \mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_K \} \) is the set of cluster centers.

A. Serial k-means Algorithm

The serial k-means algorithm is commonly measured by any of intra-cluster or inter-cluster criterion. A typical intra-cluster criterion is the squared-error criterion (Equation 1). It is the most commonly used and a good measure of the within-cluster variation across all the partitions. For the current work, we consider intra-cluster squared-error function to evaluate the present scheme of clustering. In basic k-means algorithm, a set \( \mathbf{D} \) of \( d \)-dimensional data is partitioned into \( K \) clusters, starting with a set of \( K \) randomly generated initial center vectors. The process iterates through the following steps:

- assignment of data to representative centers upon minimum distance, and
- computation of new cluster centers.

The process stops when cluster centers (or the metric \( \mathcal{M} \)) become stable for two consecutive iterations. The time complexity of the k-means clustering algorithm is \( O(TKN) \) where \( N \) is the number of input patterns, \( K \) is the desired number of clusters and \( T \) is the number of iterations needed to complete the clustering. The high time complexity makes k-means unsuitable for large applications. Being a greedy algorithm, it often converges to a local minima, may produce empty clusters, but performs well enough for smaller data sets.

B. Existing parallel k-means algorithms

To make k-means applicable for large databases, parallel implementations have been attempted. Most of these methods use a master-slave network, or an SIMD or Systolic architectures [7-10]. Since these algorithms use significant amount of data transfers among the processors they become time consuming processes. In the following, we describe two parallel versions of the k-means.

Kantabutra and Couch algorithm:

This parallel k-means algorithm [7] is executed on a network of processors connected in a master-slave fashion. The master processor is responsible for initial distribution of data among the slave processors. After clustering in the slaves, the master collects the output clusters.

An initial center vector \( \mathbf{z}_i \) is generated at each slave \( (P_i) \) independently. Each slave broadcasts its own center to the other slaves. Then each slave allocates local data items to the nearest center vectors (forming local clusters). After this all local data items from a slave are transferred to the other slaves, and the already created local clusters are augmented by collection of data items transmitted from other slaves on the basis of minimum distance. New values of center vector \( \mathbf{z}_i \) at each slave \( (P_i) \) are now computed by averaging. At this time the first iteration is completed. At the next step, each slave broadcasts its own center vector to the other slaves again, and the next iteration starts. The process terminates when the center vectors \( \mathbf{z}_i \) at each processor \( (P_i) \) become stable for two consecutive iterations.

This algorithm uses a number of broadcast of the center vectors as well as huge amount of data transfers among the processors during every iteration. It is shown by the authors that the communication overhead is a bottleneck of their algorithm, specially when data size is large.

Dhillon and Modha algorithm:

A better parallel version of the k-means is suggested by Dhillon and Modha in [16]. Instead of transferring all local data items during each iteration, only a very limited amount of significant information are transmitted. Specifically, the items transferred are the sum of all data elements that belong to a local sub-cluster, the number of elements in each local sub-cluster and the local sum of within cluster variation (MSE, minimum squared error). Here, one of the processors generates a set of initial cluster centers and broadcasts it to the other processors. In each iteration, each of the processors computes the sum of locally clustered data patterns and maintains a count of the number of items in each local cluster. The squared error value for the local clusters is also computed. Each processor broadcasts its local sum, count and squared error values to all the others. Center vectors for the next iteration are generated at each processor by dividing the total of sum values by the total of count values. A total of squared error values is also computed, at each site, as a measure of the stopping criteria.

Although the above algorithm reduces the communication overhead significantly, it transfers some local measures, from each site, other than original data items which may not be agreeable at certain situations for security and privacy reasons. Also, it is a better idea to limit the data transfers to the local center vectors only.

III. PROPOSED SCHEME FOR DISTRIBUTED CLUSTERING

The clustering schemes employing the parallel k-means algorithm (Section II) requires a significant amount of data transfer in each iteration. The amount of data transfer can be significantly reduced if only the local center vectors, instead of the whole subset of local data, are transferred. However, execution of the conventional
**A. Basis of the proposed scheme**

In the basic k-means algorithm, an iteration starts with a set of old center vectors \( z^{(n)} \), the data elements are distributed to corresponding clusters depending on minimum Euclidean distance, and then a set of new cluster centers \( z^{(n+1)} \) is generated by averaging the data elements. This center updatation procedure can be mathematically described as

\[
Z_k^{(n)} = \frac{1}{n_k} \left\{ \sum_{x_j \in C_i} \| x_j \| + z_i^{(n)} \right\}
\]

(2)

where, \( n_k \) is the number of elements in cluster \( C_k \). If the new centers \( Z_k^{(n)} \) do not match exactly with the old centers \( z_i^{(n)} \), the k-means algorithm enters into the next iteration assuming \( Z_k^{(n)} \) as \( z_i^{(n)} \). In case of parallel or distributed clustering, the data set is divided or pre-distributed randomly among a set of parallel processors. During each iteration, each processor tries to update its own centers based on local data and then shares its achievements with other processors while merging local centers to generate a set of global centers. In practice, the processors compute global centers by averaging local centers that correspond to the same global cluster. The global centers are then used for the next iteration of the parallel process. However, this strategy is good as long as none of the processors produce an empty or null sub-cluster. Since a null sub-cluster does not posses a defined center vector, generation of global average center vectors is not possible.

In our proposed distributed clustering scheme, we deny the formation of such null sub-clusters [18] and consider \( Z_k^{(n)} \) as a data member of the concerned sub-cluster while computing the new centers. Therefore,

\[
Z_k^{(n)} = \frac{1}{n_k} \left\{ \sum_{x_j \in C_i} \| x_j \| + z_i^{(n)} \right\}
\]

(3)

Equation 3 denotes that every sub-cluster, in a processor, should have at least one element. Incorporation of \( Z_k^{(n)} \) in computation of \( Z_i^{(n)} \), may affect the convergence of cluster centers and the quality of clustering solutions. However, since the value of \( Z_i^{(n)} \) soon becomes well inside the concerned sub-cluster (within a few iterations), the effect of \( Z_i^{(n)} \) will be very insignificant and for a large data set it can be assumed to be negligible. The following subsection reports the design of the proposed distributed algorithm, referred to as distributed \( m \)-k-means algorithm, for clustering of large and distributed data sets.

Note that, Equation 3 can also be used in case of the serial k-means algorithm to produce similar outputs as that of the original k-means algorithm. With the suggested modification (in the center computation step), we call it the serial \( m \)-k-means algorithm.

**B. The distributed \( m \)-k-means algorithm**

The execution steps of the serial \( m \)-k-means algorithm to form clusters are essentially similar to those of the serial k-means algorithm. The processor \( P \) maintains the cluster structures in its own local memory and iterates through the steps of the \( m \)-k-means algorithm to evaluate a final set of cluster centers \( Z \).

In a distributed environment, with a network of processors, the data set \( D \) is assumed to be distributed among the processors. These processors execute the \( m \)-k-means parallely. Figure 1 illustrates the execution of distributed \( m \)-k-means algorithm in a system with 3 processors and the data set is assumed to have \( K = 3 \) clusters.

![Figure 1](image-url)

**Input:** a set \( D \) of \( d \)-dimensional data and an integer \( K \).

**Output:** \( K \) clusters

**begin**
for each processor \( P \) \( do \) in parallel begin
randomly initialize \( K \) center vectors \( z_i \) for \( 1 \leq k \leq K \);
repeat
    distribute \( x_j \) for \( 1 \leq j \leq n \) to clusters \( C_i \) for \( 1 \leq k \leq K \) upon minimum distance \( d_k = ||x_j - z_k||^2 \)
    compute \( z_k^{(n)} \) according to Equation 3, for all \( k \);
end for
broadcast $z_k^{(s)}$ for $1 \leq k \leq K$ to other processors; compute average of own center and those sent from other processors and replace old centers as
$$z_k^{(s+1)} = \frac{1}{N_o} \sum_{j=1}^{N_o} z_j^{(s)}$$
for $j = 1 \ldots N_o$ of Processors

until center vectors are stable
end end

The above algorithm reveals that the proposed distributed clustering scheme can be run in any number of processors, and no transfer of actual pattern vectors (local data) is required between the processors. Only the center vectors are exchanged for updation. After execution each processor holds its own data clustered around the global center vectors. In the following sections, we shall try to prove that the serial and distributed $m_k$-means algorithms converge to $k$-means centers and the rate of convergence is almost equal to that of the serial $k$-means for large data sets.

C. Proof of convergence for serial $m_k$-means

As mentioned in Section III-A, incorporation of $z_k^{(s)}$ in computing $z_k^{(s+1)}$ may affect the rate of convergence of the proposed algorithm. However, this effect is insignificant and for a large data set it can be assumed to be negligible. An analytical measure is described in this subsection to estimate the effect of the old centers while computing new ones.

In light of the fact that old centers are used to compute new centers, the $m_k$-means may take some more iterations, for convergence of cluster centers, compared to the $k$-means algorithm. It can be experimentally verified that $m_k$-means continues for a few iterations even after all data items reached their destination clusters (i.e., there is no more inter-cluster data transition). Therefore, we can assume two phases of convergence in the $m_k$-means algorithm: (i) convergence for the cluster structures (when data transition is complete) and (ii) convergence for final cluster centers. The number of iterations taken to distribute all the data elements to their respective clusters is found almost the same as that of the $k$-means algorithm, whereas convergence of cluster centers may require some more iterations in $m_k$-means.

Let us consider that at iteration $t$, data items are distributed among correct clusters, but center vectors are yet to be stable. Let, at this stage, a certain cluster be represented by $C = \{x_1, x_2, \ldots, x_m\}$ and the corresponding center is $z(t) = x$ (say), where
$$x = \frac{x_1 + x_2 + \cdots + x_m}{m}$$
due to the initial center effect. Therefore, the following is a sequence of center vectors obtained in the subsequent iterations:

$$z(t+1) = \frac{1}{m+1} \sum_{i=1}^{m} x_i + z(t)$$
$$z(t+2) = \frac{1}{m+1} \sum_{i=1}^{m} x_i + \frac{1}{m+1} \sum_{i=m+1}^{m+n} x_i + \frac{x}{m+1}$$

Proceeding in a similar fashion, we get

$$z(t+n) \approx \frac{\sum_{i=1}^{m} x_i}{m+1} + \frac{\sum_{i=m+1}^{m+n} x_i}{m+1}$$

For large values of $m$ or $u$,

$$z(t+n) \approx \frac{\sum_{i=1}^{m} x_i}{m+1} + \frac{\sum_{i=m+1}^{m+n} x_i}{m+1}$$

which is the desired cluster center (using $m_k$-means). The additional $u$ iterations needed for final convergence. For large data sets (large $m$), $u$ should be small. That is, the effect of initial centers toward convergence is negligible.

D. Proof of convergence for distributed $m_k$-means

The above convergence theory is developed for the serial version of the $m_k$-means algorithm. In the following we show that this condition is equally valid for the distributed algorithm. Initially we shall consider two specific situations and then a general proof is tried.

Distributed execution with non-empty sub-clusters:

Let $D = \{x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n\}$ be a data set with $(m+n)$ data elements forming two well separated clusters $X$ and $Y$ as shown in Figure 2. If we apply serial $k$-means algorithm to partition this data set into two clusters, we get the following center vectors

$$z_1 = x_1 + x_2 + \cdots + x_m \text{ and } z_2 = y_1 + y_2 + \cdots + y_n$$

Figure 2. A simple 2-dimensional data and corresponding clusters.

Now let us divide the data set $D$ into $D_1$ and $D_2$ for two different processors $P_1$ and $P_2$ such that $D = D_1 \cup D_2$. Due to unknown distribution of data in $D$, in general, it is expected that each of $D_1$ and $D_2$ contains elements from $X$ and $Y$ both. Without loss of generality, for ease of explanation we consider that both $X$ and $Y$ have two sub-clusters $X_1$, $X_2$ and $Y_1$, $Y_2$ respectively (refer Figure 2), where (say)

$$X_1 = \{x_1, x_2, \ldots, x_p\}, \quad X_2 = \{x_{p+1}, x_{p+2}, \ldots, x_m\}$$
$$Y_1 = \{y_1, y_2, \ldots, y_q\}, \quad Y_2 = \{y_{q+1}, y_{q+2}, \ldots, y_n\}$$

Let the initial distribution of data in $D_1$ and $D_2$ be such that $D_1 = X_1 \cup Y_1$ and $D_2 = X_2 \cup Y_1$. While partitioning $D_1$ and $D_2$ (in processors $P_1$ and $P_2$, say), applying the distributed algorithm, into two clusters, let us assume that at iteration $t$ the cluster structures are found in individual processors and the global center vectors are $x$ and $y$ respectively. The center vectors in the individual processors ($P_1$ and $P_2$) at the next iteration are, therefore,
\[
z_{11} = \frac{x_1 + x_2 + \cdots + x_p + x}{p + 1}, \quad z_{12} = \frac{y_{q1} + y_{q2} + \cdots + y_q + y}{n - q + 1}
\]
\[
z_{21} = \frac{x_{m+1} + x_{m+2} + \cdots + x_n + x}{m - p + 1}, \quad z_{22} = \frac{y_1 + y_2 + \cdots + y_q + y}{q + 1}
\]
where \( z_{ij} \) represents the center of the /th sub-cluster in the \( i \)/th processor. Averaging for the global centers, we arrive at
\[
z_1 = \frac{y_{p+1} + y_{q+1} + \cdots + y_q + y + y_1 + y_2 + \cdots + y_q + y}{2(q + 1)}
\]
\[
z_i = \frac{y_{p+1} + y_{q+1} + \cdots + y_q + y + y_1 + y_2 + \cdots + y_q + y}{q + 1}
\]
Further, if we assume that \( m = 2p \) and \( n = 2q \) (considering even values of \( m, n \) for ease of explanation), the above centers become
\[
z_i = \frac{x_1 + x_2 + \cdots + x_p + x + x_{p+1} + x_{p+2} + \cdots + x_n + x}{m + 2}
\]
\[
z_2 = \frac{y_1 + y_2 + \cdots + y_q + y + y_1 + y_2 + \cdots + y_q + y}{n + 2}
\]
Now proceeding as in the case of serial \( m \)-means, we get
\[
z_i^{(t+1)} = x
\]
\[
z_i^{(t+1)} = \frac{1}{m + 2} \sum_{t=1}^{m} x_i + \frac{2}{m + 2} x
\]
\[
z_i^{(t+2)} = \frac{1}{m + 2} \sum_{t=1}^{m} x_i + \frac{2}{m + 2} \sum_{t=1}^{m} x_i + \frac{4}{m + 2} x
\]
Proceding in a similar fashion, we get
\[
z_i^{(t+u)} = \frac{1}{m + 2} \sum_{t=1}^{m} x_i + \frac{2}{m + 2} x
\]
For large values of \( m \) or \( u \),
\[
z_i^{(t+u)} = \frac{1}{m + 2} \sum_{t=1}^{m} x_i + \frac{2}{m + 2} x
\]
A similar result will be obtained for \( z_2 \) also. The above centers \( z_1 \) and \( z_2 \) are exactly the same as that obtained \( (z_i^{(t+u)}) \) at iteration number \( t+u \) while running the serial \( m \)-means algorithm. Hence, it can be said that, both of these centers will converge to the desired centers at iteration number \( t+u \).

**Distributed execution with empty sub-clusters:**

We can extend the concept of convergence in case of a system with empty sub-clusters. Let, the initial distribution of data in \( D_1 \) and \( D_2 \) be such that,
\[
D_1 = X_1 \cup X_2 = \{x_1, x_2, \ldots, x_m\}
\]
\[
D_2 = Y_1 \cup Y_2 = \{y_1, y_2, \ldots, y_n\}
\]
and after data distribution is complete, at iteration \( t \), the center vectors are \( x \) and \( y \) respectively.

Assuming each of the processors \( P_1 \) and \( P_2 \) have an empty cluster, the structure of the clusters are
\[
C_{11} = \{x_1, x_2, \ldots, x_m\}, \quad C_{12} = \{\Phi\}
\]
\[
C_{21} = \{\Phi\}, \quad C_{22} = \{y_1, y_2, \ldots, y_n\}
\]
Therefore, at the next iteration of the distributed \( m \)-means, the sub-cluster centers are
\[
z_{11} = \frac{x_1 + x_2 + \cdots + x_m + x}{m + 1}, \quad z_{12} = y
\]
\[
z_{21} = x, \quad z_{22} = \frac{y_1 + y_2 + \cdots + y_n + y}{n + 1}
\]
The above situation of empty clusters (at iteration \( t \), after data distribution is complete) is possible only if
\[
x = x_1 + x_2 + \cdots + x_m - \frac{x}{m}, \quad y = y_1 + y_2 + \cdots + y_n - \frac{y}{n}
\]
Therefore, for such a case
\[
z_{11} = \frac{x_1 + x_2 + \cdots + x_m + x}{m + 1}, \quad z_{12} = y
\]
\[
z_{21} = x, \quad z_{22} = \frac{y_1 + y_2 + \cdots + y_n + y}{n + 1}
\]
That is,
\[
z_{11} = \frac{x_1 + x_2 + \cdots + x_m}{m}, \quad z_{12} = \frac{y_1 + y_2 + \cdots + y_n}{n}
\]
and the global averages are
\[
z_1 = \frac{z_{11} + z_{21}}{2}, \quad z_2 = \frac{z_{12} + z_{22}}{2}
\]
Therefore, the proposed distributed algorithm performs equally well irrespective of the possibility of empty sub-clusters in any of the underlying processors.

**General Proof of convergence for distributed \( m \)-means:**

We claim that the distributed \( m \)-means centers \( (z_{10}, z_{20}) \) converges to the \( k \)-means centers \( (z_{10}, z_{20}) \). A proof of the above claim is given below. It is to be noted that \( m \)-means and \( k \)-means follow essentially the same steps differing only at the center computation steps, where \( m \)-means considers an extra data item (the old center vector). These old centers gradually approach toward the actual cluster centers by virtue of averaging. [We already have shown this fact analytically for 1-processor (serial) and 2-processor (distributed) systems. One can use more number of processors and will arrive at the same result.] Hence, by considering the convergence of the distributed \( m \)-means, we can proceed as follows to get a general proof of our claim.

Let us consider a \( P \)-processor and \( K \)-cluster situation. For a particular cluster, let the local data items and the corresponding local centers in different processors, at any instant after initial distribution, are denoted by

<table>
<thead>
<tr>
<th>Proc. No.</th>
<th>Data items</th>
<th>Center vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( x_{11} ), ( x_{12} ), \ldots, ( x_{1p} )</td>
<td>( z_1 )</td>
</tr>
<tr>
<td>2</td>
<td>( x_{21} ), ( x_{22} ), \ldots, ( x_{2p} )</td>
<td>( z_2 )</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>( P )</td>
<td>( x_{p1} ), ( x_{p2} ), \ldots, ( x_{pn} )</td>
<td>( z_p )</td>
</tr>
</tbody>
</table>

respectively. In uniprocessor based serial \( k \)-means, the current center vector will be (note that here \( z_1, z_2, \ldots, z_p \) represent centers of the same cluster formed partly in different processors)
\[
z_{10} = \frac{x_1 + \cdots + x_m + x_{20} + \cdots + x_{2m} + \cdots + x_{p0} + \cdots + x_{pm}}{n_1 + n_2 + \cdots + n_p}
\]
In case of the distributed \( m \_k \)-means algorithm, we have
\[
z^{(n)} = z_1^{(n)} + z_2^{(n)} + \ldots + z_p^{(n)}
\]
\[
= \frac{1}{p} \left( x_{11}^{(n)} + \ldots + x_{1n}^{(n)} + z_{1}^{(n)} \right) + \ldots + \frac{1}{p} \left( x_{n1}^{(n)} + \ldots + x_{np}^{(n)} + z_{p}^{(n)} \right)
\]
where, \( W_i = x_{i1} + x_{i2} + \ldots + x_{in_i} \). Clearly, from above formulations, \( k \)-means centers and distributed \( m \_k \)-means centers are unequal during an intermediate iteration. However, in the final iteration (i.e., at termination), since the algorithm will ultimately converge, we have \( Z^{(n)} = \bar{Z}^{(n)} \), and also,
\[
z^{(n)} = \frac{w_1 + z^{(n)}}{n_1 + 1} + \ldots + \frac{w_p + z^{(n)}}{n_p + 1}
\]
in all the processors. Therefore, we can write,
\[
z_{in\_sum} = \frac{w_1}{n_1} + \ldots + \frac{w_p}{n_p}
\]
IV. COMPLEXITY ANALYSIS

There are two parts in the complexity of any parallel and distributed algorithm: Computation complexity \( t_{COMP} \) and Communication complexity \( t_{COMM} \). During computation, we compute the distance of each data from the corresponding center vectors (this is the major computing step), and during communication phase, we transfer data, center vectors and other relevant information from one processor to others (in broadcast mode).

Let us consider the complexity of one iteration of the distributed \( m \_k \)-means algorithm. Let, \( t_{data} \) be the actual communication time for a single data item, and \( t_{startup} \) be the time needed to establish a connection. Within one iteration we perform parallel data (\( K \) center vectors only) transfer once. Therefore, the complexity (per iteration) is
\[
t_{comm} = t_{startup} + K \times t_{data}
\]
Similarly, the time needed for distance computation becomes, \( t_{comp} = K \times n \times t_{dist} \) where, \( T_{dist} \) is the time needed to compute a single distance, and \( n = N/P \). Let, \( T \) be the total number of iterations of the \( k \)-means loop. Then, the overall complexities become
\[
t_{COMM} = T \times \{ t_{startup} + K \times t_{data} \}
\]
\[
t_{COMP} = T \times K \times n \times t_{dist}
\]
It is found that a new connection can be set up in a dedicated network almost instantly. Hence, if we ignore the communication startup overhead, the complexities become,
\[
t_{COMM} = T \times K \times t_{data} \quad \text{and} \quad t_{COMP} = T \times K \times n \times t_{dist}
\]
In our system, \( P \) distance computations and also \( P \) data transmissions can be done parallelly. Therefore, we can attain a speedup of \( \approx P \).

The distributed scheme proposed by Dhillon et. al. [10], shows the communication and computation complexities to be (using our present notations)
\[
t_{COMM} = T \times K \times t_{data} + \text{time to transfer} \ LVS, \ count, \ MSE \quad \text{and} \quad t_{COMP} = T \times K \times n \times t_{dist} + \text{time to compute} \ LVS, \ count, \ MSE
\]
where, \( LVS \) is the sum of all local vectors in individual clusters, \( count \) is the number of elements in individual local clusters, and \( MSE \) is Euclidean sum of minimum squared error for all local data.

For Kantabutra’s scheme [7], the communication and computation complexities come out to be (using our present notations)
\[
t_{COMM} = T \times n \times t_{data} \quad \text{and} \quad t_{COMP} = T \times K \times n \times t_{dist}
\]
For large data sets, \( K << n \). Hence, the proposed scheme requires lesser data transfer. In case of all the parallel and distributed algorithms computation of center vectors contributes to the computation complexity. Although we do not make an explicit analysis of this complexity, the actual computation time is measured and reported in Section V. A comparison of communication complexities between Dhillon’s algorithm and the present algorithm is next provided. We consider data transfer in terms of scalar components only. The number of scalar values transmitted are found to be as follows
\[
N_{Dhillon} = K \times d + K \times 1 \quad \text{and} \quad N_{Proposed} = K \times d
\]
Therefore, communication complexity ratio becomes
\[
R = \frac{N_{Dhillon}}{N_{Proposed}} = \frac{K \times d + K \times 1}{K \times d} \approx 1 + \frac{K \times 1}{K \times d} \approx 1 + \frac{1}{d}
\]
where, \( K >> 1 \). Now, for data with lower value for dimension \( d \), \( R \) has a significant effect. For \( d = 2, R = 1.5 \), i.e., in such case, Dhillon’s algorithm involves 50% more communication overhead.

V. EXPERIMENTAL RESULTS

This section provides the performance comparison of the conventional \( k \)-means, serial \( m \_k \)-means and the distributed \( m \_k \)-means in terms of rate of convergence and the quality of the solution \( M \). The efficiency of the proposed distributed \( m \_k \)-means algorithm is then compared with respect to other existing parallel and distributed algorithms.

A. Comparison of different \( k \)-means algorithms upon convergence and quality of solutions

In this subsection, we report the performances of serial and parallel \( k \)-means and distributed \( m \_k \)-means algorithms while executed on four artificial and four real-life data sets. The artificial data sets are Circular_5_2, Circular_6_2, Spherical_4_3 and Elliptical_10_2 [19-22]. The names imply the structure of underlying clusters, concatenated with the number of clusters present in the data and its dimensions. For example, in Circular_5_2 data set the clusters are circular in nature. There are five clusters and the dimension is 2. The real-life data sets are the Iris [22], Crude_Oil [23], Breast_Cancer and Color_Moments [24]. The Color_Moments data is derived from 68,040 images of the corel image features collection of the UCI KDD archive [24]. The number of clusters for the Iris, Crude_Oil and Breast_Cancer data sets are known to be 3, 3 and 2 respectively. For the Color_Moments data, we assumed the number of clusters to be 10. For the purpose of comparison, the distributed algorithm is executed in a simulated environment using 2, 3 and 5 processors.
(considering them to form a distributed network). The results of comparison are given in Table I. The Circular_5_2 and Elliptical_10_2 data sets (clustered using serial m_k-means) are shown in Figures 3 to 4.

Table I. Results of Serial and Distributed Algorithms on Data sets

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Iter</th>
<th>Serial k-means</th>
<th>Iter</th>
<th>m_k-means</th>
<th>Iter</th>
<th>Distributed m_k-means</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circular_5_2</td>
<td>6</td>
<td>327.530</td>
<td>8</td>
<td>328.437</td>
<td>2</td>
<td>328.941</td>
</tr>
<tr>
<td>Circular_6_2</td>
<td>6</td>
<td>374.541</td>
<td>8</td>
<td>374.542</td>
<td>2</td>
<td>374.941</td>
</tr>
<tr>
<td>Spherical_4_3</td>
<td>4</td>
<td>749.978</td>
<td>8</td>
<td>749.978</td>
<td>2</td>
<td>750.002</td>
</tr>
<tr>
<td>Elliptical_10_2</td>
<td>14</td>
<td>949.389</td>
<td>14</td>
<td>949.345</td>
<td>2</td>
<td>949.657</td>
</tr>
<tr>
<td>Iris</td>
<td>5</td>
<td>97.205</td>
<td>5</td>
<td>97.225</td>
<td>2</td>
<td>97.238</td>
</tr>
<tr>
<td>Crude_Oil</td>
<td>16</td>
<td>279.743</td>
<td>16</td>
<td>279.743</td>
<td>2</td>
<td>289.557</td>
</tr>
<tr>
<td>Breast_Cancer</td>
<td>3</td>
<td>2986.96</td>
<td>3</td>
<td>2986.96</td>
<td>2</td>
<td>2988.557</td>
</tr>
<tr>
<td>Color_Moments</td>
<td>152</td>
<td>133181.1</td>
<td>28</td>
<td>133181.1</td>
<td>2</td>
<td>133320.35</td>
</tr>
</tbody>
</table>

Table II. Results of Execution of Serial K-Means on 3D Data

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Iter</th>
<th>Exec time</th>
<th>Transfer time</th>
<th>Total time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D-Data-1</td>
<td>16</td>
<td>10.6</td>
<td>0.0049</td>
<td>10.6049</td>
<td>3.30</td>
</tr>
<tr>
<td>3D-Data-2</td>
<td>15</td>
<td>10.1</td>
<td>0.0054</td>
<td>10.1054</td>
<td>3.31</td>
</tr>
<tr>
<td>3D-Data-3</td>
<td>12</td>
<td>18.5</td>
<td>0.0083</td>
<td>18.5083</td>
<td>2.02</td>
</tr>
<tr>
<td>3D-Data-4</td>
<td>14</td>
<td>25.1</td>
<td>0.0145</td>
<td>25.1145</td>
<td>1.95</td>
</tr>
<tr>
<td>3D-Data-5</td>
<td>16</td>
<td>33.5</td>
<td>0.0372</td>
<td>33.50372</td>
<td>1.82</td>
</tr>
<tr>
<td>3D-Data-6</td>
<td>50</td>
<td>998.5</td>
<td>0.0165</td>
<td>998.50165</td>
<td>1.11</td>
</tr>
<tr>
<td>3D-Data-7</td>
<td>50</td>
<td>10.8</td>
<td>0.0033</td>
<td>10.8033</td>
<td>3.31</td>
</tr>
<tr>
<td>3D-Data-8</td>
<td>50</td>
<td>10.8</td>
<td>0.0033</td>
<td>10.8033</td>
<td>3.31</td>
</tr>
</tbody>
</table>

Table III. Results of Kantabutra’s Algorithm on 3D Data

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Iter</th>
<th>Exec time</th>
<th>Transfer time</th>
<th>Total time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D-Data-1</td>
<td>36</td>
<td>10.1</td>
<td>0.0064</td>
<td>10.1064</td>
<td>2.01</td>
</tr>
<tr>
<td>3D-Data-2</td>
<td>33</td>
<td>10.1</td>
<td>0.0064</td>
<td>10.1064</td>
<td>2.01</td>
</tr>
<tr>
<td>3D-Data-3</td>
<td>31</td>
<td>18.5</td>
<td>0.0083</td>
<td>18.5083</td>
<td>2.02</td>
</tr>
<tr>
<td>3D-Data-4</td>
<td>15</td>
<td>25.1</td>
<td>0.0145</td>
<td>25.1145</td>
<td>1.95</td>
</tr>
<tr>
<td>3D-Data-5</td>
<td>46</td>
<td>33.5</td>
<td>0.0372</td>
<td>33.50372</td>
<td>1.82</td>
</tr>
<tr>
<td>3D-Data-6</td>
<td>50</td>
<td>998.5</td>
<td>0.0165</td>
<td>998.50165</td>
<td>1.11</td>
</tr>
<tr>
<td>3D-Data-7</td>
<td>50</td>
<td>10.8</td>
<td>0.0033</td>
<td>10.8033</td>
<td>3.31</td>
</tr>
<tr>
<td>3D-Data-8</td>
<td>50</td>
<td>10.8</td>
<td>0.0033</td>
<td>10.8033</td>
<td>3.31</td>
</tr>
</tbody>
</table>

Table IV. Results of Dhillon’s Algorithm on 3D Data

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Iter</th>
<th>Exec time</th>
<th>Transfer time</th>
<th>Total time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D-Data-1</td>
<td>37</td>
<td>10.6</td>
<td>0.0049</td>
<td>10.6049</td>
<td>3.30</td>
</tr>
<tr>
<td>3D-Data-2</td>
<td>37</td>
<td>10.1</td>
<td>0.0054</td>
<td>10.1054</td>
<td>3.31</td>
</tr>
<tr>
<td>3D-Data-3</td>
<td>42</td>
<td>121.4</td>
<td>0.00737</td>
<td>121.40737</td>
<td>3.06</td>
</tr>
<tr>
<td>3D-Data-4</td>
<td>56</td>
<td>326.8</td>
<td>0.00983</td>
<td>326.80983</td>
<td>2.36</td>
</tr>
<tr>
<td>3D-Data-5</td>
<td>46</td>
<td>33.5</td>
<td>0.0372</td>
<td>33.50372</td>
<td>1.82</td>
</tr>
<tr>
<td>3D-Data-6</td>
<td>55</td>
<td>994.3</td>
<td>0.00965</td>
<td>994.30965</td>
<td>2.62</td>
</tr>
<tr>
<td>3D-Data-7</td>
<td>55</td>
<td>1034.1</td>
<td>0.01018</td>
<td>1034.1018</td>
<td>2.28</td>
</tr>
<tr>
<td>3D-Data-8</td>
<td>50</td>
<td>1036.9</td>
<td>0.01053</td>
<td>1036.9053</td>
<td>2.55</td>
</tr>
</tbody>
</table>

Table V. Results of Present Algorithm on 3D Data

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Iter</th>
<th>Exec time</th>
<th>Transfer time</th>
<th>Total time</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D-Data-1</td>
<td>37</td>
<td>8.6</td>
<td>0.00455</td>
<td>8.60455</td>
<td>4.07</td>
</tr>
<tr>
<td>3D-Data-2</td>
<td>38</td>
<td>10.0</td>
<td>0.00544</td>
<td>10.00544</td>
<td>3.33</td>
</tr>
<tr>
<td>3D-Data-3</td>
<td>47</td>
<td>124.1</td>
<td>0.00737</td>
<td>124.10737</td>
<td>3.06</td>
</tr>
<tr>
<td>3D-Data-4</td>
<td>60</td>
<td>330.3</td>
<td>0.00983</td>
<td>330.30983</td>
<td>2.36</td>
</tr>
<tr>
<td>3D-Data-5</td>
<td>54</td>
<td>630.5</td>
<td>0.00930</td>
<td>630.50930</td>
<td>2.71</td>
</tr>
<tr>
<td>3D-Data-6</td>
<td>55</td>
<td>964.3</td>
<td>0.00965</td>
<td>964.30965</td>
<td>2.62</td>
</tr>
<tr>
<td>3D-Data-7</td>
<td>55</td>
<td>1337.3</td>
<td>0.01018</td>
<td>1337.3018</td>
<td>2.28</td>
</tr>
<tr>
<td>3D-Data-8</td>
<td>55</td>
<td>1567.2</td>
<td>0.01053</td>
<td>1567.2053</td>
<td>2.55</td>
</tr>
</tbody>
</table>
almost similar to those required for the serial algorithm. The distributed Two different categories of artificial data sets have the same dimensionality and the same number of clusters. We have maintained for experimentation, we have used large artificially determined the performances of these algorithms. Hence, for the distributed algorithm also, we maintained 3 processors. We measured computation and communication times in units of $10^4$ clock ticks. We observed that to send one scalar data, we need only $0.0000135 \times 10^4$ clock ticks. This is very small compared to computation times.

Execution time for clustering 3D artificial data sets using serial $k$-means, Kantabutra’s, Dhillon’s and the proposed distributed $m_k$-means algorithms are shown in Tables II to V. The obtained speedup are also noted in these tables. It is observed from these tables that, the proposed distributed $m_k$-means algorithm provides significantly higher speedup than the parallel $k$-means algorithm. The graphs of Figure 5 show the total execution times of different algorithms against data size. For fair comparison we have used 3 processors in all the cases. If the number of processors is increased, much higher speedup can be obtained (depending on the number of processors used) with the proposed distributed algorithm. The results of the run of distributed algorithm for the Breast Cancer (BC) and Color Moments (CM) data sets, by varying the number of processors are reported in Table VII. The variation of speedup against the number of processors used are shown in Figure 10. From these results, it is evident that the distributed algorithm provides higher speedup for larger data sets.

### Table VI. Distribution of Iris Data in 3 Processors

<table>
<thead>
<tr>
<th>Processor Number</th>
<th>Number of elements in sub-clusters</th>
<th>Sub-clus-1</th>
<th>Sub-clus-2</th>
<th>Sub-clus-3</th>
<th>Total no. of elm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>12</td>
<td>19</td>
<td>19</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>$P_2$</td>
<td>12</td>
<td>26</td>
<td>12</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>$P_3$</td>
<td>14</td>
<td>17</td>
<td>19</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>$P_1+P_2+P_3$</td>
<td>38</td>
<td>62</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

#### B. Comparison between Parallel and Distributed $k$-means algorithms for rate of convergence and speedup

The performance of the proposed distributed $m_k$-means algorithm and an existing parallel $k$-means algorithm ([8]) over the serial $k$-means algorithm, in terms of rate of convergence and speedup, is evaluated in this subsection. Since, either in parallel or distributed processes, data or center vectors need to be communicated among the processors during each iteration, data transmission time takes a crucial role in determining the performances of these algorithms. Hence, for experimentation, we have used large artificially created data sets of different sizes. We have maintained the same dimensionality and the same number of clusters. Two different categories of artificial data data sets have been used. For each category, 8 different data sets $K = 3$ clusters and varying sizes are used. In the second instance, other 8 data sets are used. The dimensions of the data in the two categories are $d = 2$ and 3 respectively. The parallel $k$-means algorithm, used for comparison, can only be executed when the number of processors is equal to the number of clusters. Hence, for the distributed algorithm also, we maintained 3 processors.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Serial $m_k$-means</th>
<th>Distri. $m_k$-means</th>
<th>Speedup obtained</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Proc.</td>
<td>Execution Time</td>
<td>No. of Proc.</td>
<td>Execution Time</td>
</tr>
<tr>
<td>BC</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.6719</td>
<td>1.488</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.4019</td>
<td>2.488</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>0.3019</td>
<td>3.312</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.2017</td>
<td>4.958</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>0.1719</td>
<td>5.817</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0.1322</td>
<td>7.564</td>
</tr>
<tr>
<td>CM</td>
<td>1</td>
<td>12241</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3999.1494</td>
<td>3.061</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3023.1761</td>
<td>4.049</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2377.2102</td>
<td>5.149</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1779.2248</td>
<td>6.880</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>1487.2260</td>
<td>8.231</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>953.1665</td>
<td>12.842</td>
</tr>
</tbody>
</table>
VI. CONCLUSIONS

This paper proposes a distributed clustering algorithm for a network of processors that incurs very low computation and communication costs, and simultaneously ensures privacy and security of local data. The scheme is developed based on a modified version of the basic serial $k$-means algorithm. It is shown that the proposed algorithm is semantically equivalent to the serial $k$-means algorithm. A comparison of the proposed algorithm with two other existing algorithms is reported. Experimental results show that the proposed clustering scheme is much faster and can provide higher degree of data security than the others.

REFERENCES