Multi-Party \textit{k}-Means Clustering with Privacy Consideration

Teng-Kai Yu*, D.T. Lee**, Shib-Ming Chang
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan
tkyu@ntu.edu.tw, dtlee@csie.ntu.edu.tw, r97922124@ntu.edu.tw

Justin Zhan
National Center for the Protection of the Financial Infrastructure
Dakota State University, Madison, South Dakota, USA
justinzhan@gmail.com

Abstract—The \textit{k}-means clustering algorithm is a widely used scheme to solve the clustering problem which classifies a given set of \(n\) data points in \(m\)-dimensional space into \(k\) clusters, whose centers are obtained by the centroids of the points in the same cluster. The problem with privacy consideration has been studied, when the data is distributed among different parties and the privacy of the distributed data is to be preserved. In this paper, we apply the concept of parallel computing to solve the privacy-preserving multi-party \textit{k}-means clustering problem, when the data is vertically partitioned and horizontally partitioned respectively among different parties. We present algorithms for solving the problems for these two data partition models that run in \(O(nk)\) time and in \(O(mk + \log(n/k))\) time respectively. The time complexities of the algorithms are much better than others without parallel computing.

I. INTRODUCTION

In the modern ages, the usage of internet has become more and more popular. Many kinds of information are transported through the internet in huge amounts on a daily basis, such as personal medical records, credit card numbers, business transactions, etc. When dealing with this sensitive information, the privacy issues become major concerns, as any leakage or compromise of data may result in potential harm to individuals or financial losses to corporates.

Clustering is a process of grouping a set of objects into classes or clusters so that objects within a cluster are similar in comparison with one another, but are dissimilar to objects in other clusters [18]. In other words, clustering is to partition a set of data into groups according to some similarity measures of the data. It is widely used in the applications of financial affairs, marketing, insurance, medicine, chemistry, machine learning, data mining, etc. In general, the data objects with multiple attributes can be considered as points in \(m\)-dimensional space for some integer \(m > 1\), and the \(k\)-cluster problem is formally defined as follows. Given a set of \(n\) points in \(\mathbb{R}^m\), find a set of \(k\) centers \(\{c_1, c_2, \ldots, c_k\}\) so as to minimize the sum of the squares of the Euclidean distances of each point to its nearest center. It was shown in [2] [7] [10] that the \(k\)-cluster problem is NP-hard when the dimension \(m\) is part of the input, even for \(k = 2\). And only recently it has been shown that the \(k\)-cluster problem for a set of \(n\) points in the plane [27], i.e. \(m = 2\) is also NP-hard. If both \(k\) and \(m\) are fixed, the problem can be solved exactly in polynomial time [19].

The \(k\)-cluster problem being NP-hard, approximation or heuristic algorithms have been proposed [3] [8] [11] [12] [23] [24] [26] [28]. Among them the \textit{k}-means clustering algorithm [11] [26] is a simple but widely used heuristic algorithm to the \(k\)-cluster problem.

In the original problem it is assumed that the set of data for the \(k\)-cluster problem is centralized. The \textit{k}-means clustering algorithm [11] [26], shown in Section II-A, is quite straightforward. When the data is distributed among many parties, [9] gave a parallel algorithm to solve it. But if the parties who own the data want to preserve their private data, a solution to the privacy-preserving \(k\)-clustering problem becomes less obvious. How we conduct clustering and also preserve private data is an interesting problem. Depending on the problems we deal with and the ways in which the data is distributed among multiple parties, the privacy issues will arise in different situations. These kinds of problems have been studied in the field of privacy-preserving data mining or PPDM for short [1] [25].

Many research works studied the privacy-preserving of \textit{k}-means clustering algorithms, and considered various data partition models: vertically partitioned data [31], horizontally partitioned data [22], and arbitrarily partitioned data [5] [21]. We will introduce in the paper the notion of parallel computing into the privacy-preserving multi-party \textit{k}-means clustering problem and give efficient algorithms for vertically and horizontally partitioned data models respectively. As expected, the parallel algorithms are much faster than those without parallelism. We show that the speedup of these parallel algorithms is optimal for vertically partitioned data model and is nearly optimal for horizontally partitioned data model.

We summarize our contributions of this paper as follows. Let \(n\) be the number of points, \(m\) the dimensionality or the
number of attributes of data points, and $k$ the number of clusters.

- We introduce the notion of parallelism into privacy-preserving multi-party $k$-means clustering problem for the following two data partition models.
- For vertically partitioned data, we present a parallel algorithm that runs in $O(nk)$ time, when the number of parties (or processors) is $m$.
- For horizontally partitioned data, we present a parallel algorithm that runs in $O(m(k+\log(n/k)))$ time, when the number of parties (or processors) is $n$.

In Section II, we will introduce some preliminaries. In Section III, we will deal with the privacy-preserving multi-party $k$-means clustering problem for vertically partitioned data and horizontally partitioned data. We conclude in Section IV with some discussions and suggestions of future work.

II. Preliminaries

In this section, we will present some preliminaries. We first introduce the $k$-means clustering algorithm in Section II-A. We then explain our communication model in Section II-B. In Section II-C, we will introduce homomorphic encryption. Based on the homomorphic encryption, the scheme of secure add and compare will be presented in Section II-D. In Section II-E we present the single round robin method that will be used for data exchange and communication.

A. $k$-means clustering algorithm

Recall that we are given a set of $n$ data points in $\mathbb{R}^m$, $m > 1$, and want to partition the set of points into $k$ clusters so as to minimize a certain objective function. Since the general problem is NP-hard, heuristics or approximation algorithms are considered. We shall focus on the $k$-means clustering algorithm [11] [26] which is one of the most often used heuristic algorithm for the $k$-cluster problem. We only address the heuristic algorithm itself with an emphasis on its efficiency, and do not consider its performance with respect to the optimal solution. Without loss of generality we shall assume that the number $n$ of data points is significantly larger than the number $m$ of attributes and the output size $k$, the number of clusters. The $k$-means clustering algorithm [11] [26] is summarized below.

**Algorithm 1: $k$-means clustering.**

**Input:** $n$ points in $\mathbb{R}^m$, $m > 1$.

**Output:** $k$ centers, each of which is the centroid of points belonging to the same cluster.

1. Arbitrarily select a set of $k$ centers $C = \{c_1, c_2, \ldots, c_k\}$.
2. For every point, find its closest center $c_i$ and assign it to cluster $i$.
3. Based on the cluster assignment, calculate the new set of centers $C' = \{c'_1, c'_2, \ldots, c'_k\}$ for all clusters.
4. If the difference between $C$ and $C'$ is small enough, then we terminate the algorithm and output $C$.
5. If not, replace $C$ by $C'$ and goto Step 2.

In short, we arbitrarily choose the initial positions of these $k$ centers $C = \{c_1, c_2, \ldots, c_k\}$ in Step 1. In Steps 2 to 5, we first assign each point to the cluster whose center is closest to it, where we use Euclidean distance as the distance function. We then in Step 3 re-calculate a new set of $k$ clusters centered at $C' = \{c'_1, c'_2, \ldots, c'_k\}$ based on the cluster assignment in Step 2. The new center of each cluster is calculated by using the arithmetic mean method, i.e. by computing the arithmetic mean of the points in the cluster. We decide if the old set of centers is good enough by checking in Step 4 if the difference of new set of centers $C'$ and the old set of centers $C$ is small enough within a given threshold. If so, we terminate the algorithm and return $C$ as the final result. Otherwise, in Step 5, we will use the new set of centers $C'$ and iterate the process. The computation of distance between two sets of centers is not our concern of this paper, so we can adopt any method.

Due to privacy concerns, the data used to compute the cluster centers may not be revealed if they belong to different parties. Depending on the data distribution models, the operations specified in Steps 2, 3 and 4 will be handled differently. We will discuss the details later.

B. Communication model

Let us first describe the communication model that we assume in this paper as follows:

- For each pair of parties there exists a connection between them, i.e. the parties form a complete graph.
- Every party can work independently in parallel.
- In each unit of time, a party can do only one of the three operations: (1) broadcast one package of data to all other parties; (2) send one package of data to another party; (3) receive one package of data from another party.
- In each unit of time, if a party broadcasts data, all other parties will be occupied to receive that data.

The size of one package of data is assumed to be a constant.

As far as privacy preservation is concerned, we assume a semi-honest adversary model, also called honest but curious adversary model [15]. We assume that a semi-honest party will correctly follow the protocol, but whenever opportunities present themselves, it will unearth the information that it is not supposed to know by the data that it has obtained. Thus if we want to prevent leakage of data, we need to transfer data carefully. The following encryption scheme is thus adopted.

C. Homomorphic encryption

Let $(G, E, D, M)$ be a homomorphic encryption scheme [29], where $G$ is an algorithm for generating keys, $E$ and $D$ are the encryption and decryption functions, and $M$ is the message space. It has the following properties:

- $(G, E, D)$ is semantically secure [16].
- $E(a_1) \times E(a_2) = E(a_1 + a_2)$, for $a_1, a_2 \in M$.
- $E(a_1)^a = E(a \cdot a_1)$, for $a_1 \in M$ and $a \in \mathbb{N}$.

The stopping condition varies. We can also terminate the algorithm by fixing the number of iterations, if the convergence rate is too slow.
We assume that the time complexities of key generation, encryption and decryption operations are all constant.

D. Secure add and compare

Based on the homomorphic encryption, we introduce the secure add and compare scheme [31] here. It is a variation of the circuit evaluation scheme due to Yao [34]. This scheme has been shown to be semantically secure [16] and will be an important primitive operation of our algorithm. For two parties, $P_1$ and $P_2$, such that $P_1$ has numbers $a_1$ and $b_1$, and $P_2$ has numbers $a_2$ and $b_2$, we want to securely find if $a_1 + a_2 < b_1 + b_2$ without revealing the following two pieces of information to the other party: (1) the numbers possessed by each party; and (2) the difference between $a_1 + a_2$ and $b_1 + b_2$.

Secure add and compare
1) $P_1$ generates an encryption-decryption key pair $(e, d)$ of one of the homomorphic encryption schemes and sends $e$ and $e(a_1 - b_1)$ to $P_2$.
2) $P_2$ calculates $e(a \cdot s) = e(a \cdot ((a_1 - b_1) + (a_2 - b_2)))$ = $(e(a_1 - b_1) \times e(a_2 - b_2))^a$ and sends it to $P_1$, where $a \in \mathbb{N}$.
3) $P_1$ decrypts the data and if $a \cdot s \geq 0 \rightarrow a_1 + a_2 \geq b_1 + b_2$; else $a_1 + a_2 < b_1 + b_2$.

It is obvious that the Secure add and compare algorithm has time complexity $O(1)$ for both computational and communication operations, and we can therefore assume that Secure add and compare operation is a primitive function which takes constant time.

We now introduce another primitive function that will be used later on.

E. Single round robin

The following theorem in [32] is the basis for our single round robin scheme.

- An $n$-clique, or complete graph of $n$ vertices, denoted $K_n$, can be decomposed into $n - 1$ perfect matchings when $n$ is even.

Furthermore, an easy construction method was given so that every vertex can find its matching partners of all the $n - 1$ perfect matchings of $K_n$ in linear time. Each perfect matching in $K_n$ is used to represent operations that can be performed simultaneously in parallel.

Recall that one of the basic problems to deal with is that given $n$ parties $P_1, P_2, \ldots, P_n$, every party wants to communicate with all other parties. The communication model is similar to Section II-B. We can only use send but not broadcast. We assume the number of parties to be even, for otherwise we can add a dummy party $P_{n+1}$ if $n$ is odd. In each round of the single round robin we have $n/2$ matching pairs. That is, each party in each round communicates with the party it is matched to, so that in $n - 1$ rounds each party can communicate with $n - 1$ different parties corresponding to the $n - 1$ edge-disjoint perfect matchings.

Thus, if there are $n$ parties and each has a personal data, then every party can collect all of the data owned by the other $n - 1$ parties after $n - 1$ rounds of single round robin communication.

III. Parallel Privacy-Preserving Multi-Party $k$-Means Algorithms

In the classical $k$-means clustering algorithm, the data of all the points and centers is centralized. But when the data is distributed among many parties, it requires some efforts if privacy issues need to be addressed. For example, if each of the $n$ parties, which possesses the data of one of the $n$ points, needs to preserve its data, we need to modify Step 3 of the $k$-means clustering algorithm to compute new centers.

In the $k$-means clustering algorithm, we have $n$ points in an $m$-dimensional space, where each one of them is regarded as a $1 \times m$ row vector. If we combine these $n$ row vectors together, we get an $n \times m$ matrix $M_p$, which is shown below, where $p_{i,j}$ means the $j^{th}$ entry of point $p_i$.

$$
M_p = \begin{pmatrix}
p_{1,1} & p_{1,2} & \cdots & p_{1,m} \\
p_{2,1} & p_{2,2} & \cdots & p_{2,m} \\
\vdots & \vdots & \ddots & \vdots \\
p_{n,1} & p_{n,2} & \cdots & p_{n,m}
\end{pmatrix}
$$

Similarly the $k$-centers can be represented as a $k \times m$ matrix $M_c$ as shown below, where $c_{i,j}$ means the $j^{th}$ entry of center $c_i$.

$$
M_c = \begin{pmatrix}
c_{1,1} & c_{1,2} & \cdots & c_{1,m} \\
c_{2,1} & c_{2,2} & \cdots & c_{2,m} \\
\vdots & \vdots & \ddots & \vdots \\
c_{k,1} & c_{k,2} & \cdots & c_{k,m}
\end{pmatrix}
$$

In this paper we consider two kinds of data partition models (1) vertically partitioned data, i.e. each party has a column of $M_p$ and $M_c$, and (2) horizontally partitioned data, i.e. each party has a row of $M_p$. We will introduce the notion of parallel computing to solve this problem under these two data partition models.

We will deal with vertically partitioned data in Section III-A and horizontally partitioned data in Section III-B.

A. Vertically partitioned data

In the vertically partitioned data case, we assume that there are $m$ parties, $P_1, P_2, \ldots, P_m$, and that party $P_i$ has the raw data of the $i^{th}$ column of $M_p$, and the $i^{th}$ column of $M_c$.

Since all the data is partitioned into $m$ parties, the privacy issue will arise in Steps 2 and 4 of Algorithm 1. But Step 4 is much easier than Step 2, so we will focus on Step 2. The privacy issues concerned are as follows:

1) A party cannot know the data of points possessed by other parties.
2) A party cannot know the data of centers possessed by other parties.
3) A party cannot know the distances between any points and centers.
Vaidya and Clifton [31] presented an algorithm to deal with this problem for multi-party case. They used two important functions, vectors summing to zero and secure add and compare. We will also use these two functions. Furthermore, we will introduce parallelism to speed up, assuming each party has computation capabilities and can perform computation independently following the communication model described above.

In Step 2 of Algorithm 1, we need to find the closest center for each point. We will first present an algorithm for one point and parallelize it later.

We focus on a point \( p = (p_1, p_2, \ldots, p_m) \) to find its closest center. For a party \( P_i \), it only has the entry \( p_j \) of \( p \) and the \( i^{th} \) column of \( M_C \), i.e. 
\[
\begin{pmatrix}
c_{i1} \\
c_{i2} \\
\vdots \\
c_{ik}
\end{pmatrix}
\]
Let \( x_{ji} = (p_j - c_{ji})^2 \). \( P_1 \) has the \( k \times 1 \) vector 
\[
X_1 = \begin{pmatrix}
x_{11} \\
x_{12} \\
\vdots \\
x_{1k}
\end{pmatrix}
\]
P_2 has \( X_2 \) = \[
\begin{pmatrix}
x_{21} \\
x_{22} \\
\vdots \\
x_{2k}
\end{pmatrix}
\]
and \( p_m \) has \( X_m \) = \[
\begin{pmatrix}
x_{1m} \\
x_{2m} \\
\vdots \\
x_{km}
\end{pmatrix}
\]
The distance of the point \( p \) to center \( c_j \) is the square root of \( \sum_{i=1}^{m} x_{ji} \).

If we can collect the distances of \( p \) and \( \{c_1, c_2, \ldots, c_k\} \), we can find the closest center of a point \( p \). The following is an algorithm for finding the closest center for a point \( p \).

**Algorithm 2-1:** find the closest center of point \( p \).
1. \( P_1 \) creates an encryption-decryption key pair \((e,d)\) of one of the homomorphic encryption schemes and broadcasts \( e \) to all other parties.
2. \( P_1, \ldots, P_m \) send \( e(X_1), \ldots, e(X_m) \) to \( P_2 \) respectively.
3. \( P_2 \) arbitrarily creates two \( k \times 1 \) vectors \( V_1 \) and \( V_2 \) such that \( V_1 + V_2 = 0 \).
4. \( P_2 \) sends 
\[
e(X_1) + e(X_2) + \ldots + e(X_m) + e(V_1) = e(X_1) \times e(X_2) \times \ldots \times e(X_m) \times e(V_1)
\]
to \( P_1 \).
5. \( P_1 \) and \( P_2 \) do secure add and compare on \( X_1 + X_2 + \ldots + X_m + V_1 \) and \( V_2 \) to find the center \( c \) with the smallest \( \sum_{i=1}^{m} x_{ji} \) for \( j = 1, 2, \ldots, k \).
6. \( P_1 \) broadcasts cluster \( c \) to all other parties, and every party assigns this point to cluster \( c \).

We discuss the correctness, privacy issues and time complexities of Algorithm 2-1 below. In Step 3 of Algorithm 2-1, the notion of vectors summing to zero is introduced in [31].

It is obvious that the above algorithm finds for each point \( p \) its closest center. As for the privacy issues, since only \( P_i \) knows the decryption key, the data \( e(X_i) \) for \( i = 1, \ldots, m \) sent to \( P_2 \) is semantically secure. When the data is collected at \( P_1 \), it is mixed with \( V_1 \). Since \( V_1 \) is only known to \( P_2 \), \( P_1 \) cannot learn anything from \( X_1 + X_2 + \ldots + X_m + V_1 \). Thus the first and second privacy issues considered in Section III-A are addressed. The secure add and compare scheme preserves the third privacy issue in a semantically secure way.

For computational complexity, Step 1 takes constant time to create a \((e,d)\) pair. In Step 2, encryption \( e(X_i) \) of \( X_i \) takes \( O(k) \) time for each party, but the parties can do the encryption simultaneously. In Step 3, creating summing to zero vectors also takes \( O(k) \) time. The product of \( e(X_i) \) to \( e(X_m) \) and \( e(V_i) \) can be computed in \( O(mk) \) time in Step 4. Step 5 does secure add and compare \( k \) times and takes \( O(k) \) time. The secure add and compare method was described in Section II-D, and takes constant time in both computational and communication complexity. Thus the total computational complexity is \( O(mk) \).

For communication complexity, Step 1 takes \( O(1) \) time to broadcast the encryption key. Step 2 takes \( O(mk) \) time for \( P_2 \) to receive data. Step 4 takes \( O(k) \) time to send data, Step 5 takes \( O(k) \) time to do secure add and compare, and Step 6 takes \( O(1) \) time. Thus it totally takes \( O(mk) \) communication time.

Since we have \( n \) points, the total time required to find all closest centers of the \( n \) points is \( O(nmk) \).

The above algorithm is similar to [31] and does not use the concept of parallel computing. Now we want to use parallel computing to speed up Algorithm 2-1.

Considering Step 2 of Algorithm 2-1, it takes \( O(mk) \) time to collect \( e(X_1), \ldots, e(X_m) \) for one point. For \( n \) points, it takes totally \( O(nmk) \) time. Using the concept of binary tree merging scheme where each party plays the role of aggregating the product of \( e(X_i), \ldots, e(X_m) \) in Steps 2 and 4, we can easily obtain an \( O(k \log m) \) time algorithm for collecting data for one point, resulting in a total running time of \( O(nk \log m) \) time for \( n \) points.

We will give a more efficient method to deal with Step 2 to Step 4 of Algorithm 2-1 that runs \( O(nk) \) time and deal with Step 5 Algorithm 2-1 in totally \( O(nk/m) \) time for \( n \) points, i.e. totally \( O(nk) \) time. Since we have \( m \) processors (parties), this speedup is optimal for Algorithm 2-1. The details are given below.

In Algorithm 2-1 we use \( P_1 \) and \( P_2 \) to calculate the solution. \( P_1 \) and \( P_2 \) are considered as a pair of computation partners. To make Algorithm 2-1 parallel, we let \( P_2_{i-1} \) and \( P_2_i \) be partners to each other, for \( i = 1, \ldots, \lfloor m/2 \rfloor \). If \( m \) is odd we just create a dummy party to make the total number of partners even. The party whose partner is a dummy party, is assumed to be idle in the parallel computation step. We leave out the details as this does not affect the time complexity. Without loss of generality we shall assume the number \( m \) of parties to be even, so there are exactly \( m/2 \) pairs of partners. Each of the \( m \) parties will collaborate with its partner to find the closest center for one of the \( n \) points in each iteration, and thus we totally need to run \( \lceil n/m \rceil \) iterations to find all

\(^2\)Both of the computational and communication time complexities are the same as those in [31], except that the size of a package is considered as \( b \) bits in [31], but as a constant in our case.
Thus we modify Algorithm 2-1 a little to explain how to compute in parallel. In Step 1, every party creates its own encryption-decryption key pair \((e_i, d_i)\) and broadcasts to all other parties. In Step 2, we use the single round robin method (presented in Section II-E) to exchange data, so we can collect the data needed in \(m\) rounds, where we can send in parallel specific data to a specific party in each round by the perfect matching. For party \(P_i\), let its partner be \(P_{2i-1}\) and they collaboratively compute the closest center for point \(p_i\). For each party \(P_j\), it needs to send its \(X_j\) related to point \(p_j\) to \(P_{2j-1}\) using encryption key \(e_{2j}\), as shown in Algorithm 2-2 Step 2. The symmetric operations for data sent to \(P_{2i}\) are similar. In the rest of steps, we replace \(P_1\) and \(P_2\) by \(P_{2i}\) and its partner \(P_{2i-1}\), denoted as \(P_i\) and \(P_i'\) in the following algorithm.

**Algorithm 2-2:** \(P_i\) and its partner \(P_i'\) find the closest center of point \(p_i\).

1. \(P_i\) creates an encryption-decryption key pair \((e_i, d_i)\) of one of the homomorphic encryption schemes and broadcasts \(e_i\) to all other parties.
2. For \(P_j\), send \(e_i(X_j)\) to \(P_i'\), where \(X_j\) is related to point \(p_i\).
3. \(P_i'\) arbitrarily creates two \(k \times m\) vectors \(V\) and \(V'\) such that \(V + V' = \mathbf{0}\).
4. \(P_i'\) sends \(e_i(\Sigma X_j + V) = \Pi e_i(X_j) \times e_i(V)\) to \(P_i\), where \(j = 1, \ldots, m\).
5. \(P_i\) and \(P_i'\) do secure add and compare to find the closest center \(c\) of point \(p_i\).
6. \(P_i\) broadcasts cluster \(c\) to all other parties, and every party assigns point \(p_i\) to cluster \(c\).

The correctness and privacy issues addressed are similar to Algorithm 2-1. As for time complexity we make the following argument.

For computational complexity, the analysis for Algorithm 2-1 applies, and it takes \(O(nmk)\) time for each iteration. For communication complexity, since broadcasting will occupy all parties 1 unit of time, Step 1 takes \(O(m)\) time. Step 2 can be done in \(O(nmk)\) time by using the single round robin scheme. Step 4 takes \(O(k)\) time. Step 5 takes \(O(k)\) time to perform secure add and compare, and Step 6 takes \(O(m)\) time to broadcast. The total communication complexity is thus \(O(mk)\) for each iteration.

Since we need to run \(\lceil n/m \rceil\) iterations, the total time complexity is \(O(n/m) \times O(mk) = O(nk)\) for finding the closest centers for \(n\) points in both computational and communication complexities. This has an \(O(m)\) speedup factor comparing to the sequential time complexity of Algorithm 2-1.

Having discussed Step 2 of the basic \(k\)-means clustering algorithm, Algorithm 1, we briefly discuss other steps in the following. In Step 3, since it uses arithmetic mean, every party can calculate new centers of its dimension by itself. Thus there are no privacy issues involved in Step 3. In Step 4, we add the differences together and compare it to the threshold. The differences can be calculated by the same method as Step 2 of Algorithm 2-1 within no more than \(O(mk)\) time even without parallel computing.\(^3\) The privacy issues of Step 4 are also similar to Step 2 of Algorithm 1. The updates in Step 5 can be done by each party itself. This completes the discussion. More detailed information of these three steps can be found in [31].

**Theorem 3.1:** The Privacy-Preserving Multi-party \(k\)-means Clustering Problem for vertically partitioned data of \(n\) points in \(\mathbb{R}^m\) can be solved in time \(O(nk)\).

**More discussions:** In Step 5 of Algorithm 2-1, we do secure add and compare \(k-1\) times. During the comparisons, the information about the identity of the two centers being compared will be revealed. This information revealing is not that fatal, but it still needs to be addressed, if it is a concern. We can handle this problem as follows. We can let \(P_1\) have another partner, say \(P_3\). In Step 4 before \(P_2\) sends data to \(P_1\), \(P_2\) performs a random permutation \(\pi\) on the \(k\) rows of all vectors, and then sends \(\pi(e(X_1 + X_2 + \ldots + X_m + V_1))\) to \(P_1\) and \(\pi(V_2)\) to \(P_3\). \(P_1\) and \(P_3\) do secure add and compare to find the smallest row \(i\), and send to \(P_2\) to find the closest center \(j = \pi^{-1}(i)\). Adding this extra work will not change the computational and communication complexities of Algorithm 2-1, neither Algorithm 2-2.

In this section, we assume that every party only knows the \(i^\text{th}\) column of the \(M_i\). But if we let every party know the entire \(M_i\), it will not make the problem easier. Thus whether a party knows partial or whole \(M_i\) is not an essential constraint.

In all the discussions above, we assume that there are \(m\) parties. But there may be less than \(m\) parties and each party possesses more than one column of \(M_i\). If the data is evenly partitioned, the problem can be treated similarly and can be solved in time \(O(mnk/r)\), where \(r\) is the number of parties. If the partition is uneven, the time needed will become \(O(nmk)\) with bottleneck occurring at the party which possesses the most columns \(q\) of data in Step 2 to Step 4 in Algorithm 2-1 and 2-2.

The algorithm secure add and compare presented in Section II-D can be extended to a multi-party version. Given \(n\) parties \(P_1, P_2, \ldots, P_n\), and that \(P_i\) has numbers \(a_i\) and \(b_i\), for \(i = 1, 2, \ldots, n\), we want to securely find if \(\sum_{i=1}^{n} a_i < \sum_{i=1}^{n} b_i\) without revealing the following two types of information to the other parties: (1) \(a_i, b_i, a_i + b_i, (a_i - b_i), a_i \times b_i\); (2) the difference between \(\sum a_i\) and \(\sum b_i\).

\(^3\)We can easily find an \(O(k \log m)\) parallel algorithm.
Multi-party secure add and compare

1. $P_1$ creates an encryption-decryption key pair $(e, d)$ of one of the homomorphic encryption schemes and broadcasts $e$ to all other parties, and then sends $e(a_1 - b_1)$ to $P_2$.
2. For $i = 3, 4, \ldots, n$, $P_i$ sends $e(a_i - b_i)$ to $P_2$.
3. $P_2$ calculates $e(a \cdot s) = e(a \cdot \sum_{i=1}^{n} (a_i - b_i)) = (\sum_{i=1}^{n} e(a_i - b_i))^s$ and sends it to $P_1$, where $a \in \mathbb{N}$. 
4. In $P_1$, if $\alpha \cdot s \geq 0 \rightarrow \sum_{i=1}^{n} a_i \geq \sum_{i=1}^{n} b_i$; else $\sum_{i=1}^{n} a_i < \sum_{i=1}^{n} b_i$.

It takes $O(n)$ time with the bottleneck in Step 2. If we use a tree structure merging scheme, we can accomplish it in $O(\log n)$ time. In some cases we can transfer data in parallel like in Algorithm 2-2. For the problem in this section, we can use this Multi-party secure add and compare algorithm to replace the one used in Step 2 to Step 5 of Algorithm 2-1 and 2-2. This will conduct the same time complexities.

B. Horizontally partitioned data

In this case, each party $P_i$ has the raw data of the $i$th row of $M_P$ and we assume that every party knows the whole $M_P$. Thus every party can easily find for its own data the closest center. Therefore, it will have no privacy issues at all in Step 2 of the Algorithm 1, and neither in Steps 4 and 5. The privacy issues listed below will arise in Step 3 for calculating the new centers in each iteration:

1. A party cannot know the data of points possessed by other parties.
2. A party cannot know the closest centers of points possessed by other parties.
3. A party cannot know the number of points, if any, assigned to a center for all of the centers.

Jha et al. [22] presented an encryption based algorithm to find new centers (Step 3 of Algorithm 1) for two-party case in Section 4.2 of [22]. Their algorithm cannot be directly applied to solve the multi-party case. Because $e(0)$ must be $0$ or $1$, since $e(0) \times e(0) = e(0)$, whether there is any point assigned to a center can be easily guessed during data exchange. Thus it needs extra efforts, and we will use the idea of vectors summing to zero to overcome this problem.

Similar to the case of vertically partitioned data, we first present a sequential algorithm to calculate one new center, and then give a parallel algorithm to speed up the process to find all $k$ new centers.

For any party, whether or not to assign its point to cluster $c$ can be decided by itself. For $P_i$, if its point is assigned to $c$, we assign 1 to $y_i$, and let $X_i$ be the data of the point held by $P_i$. Otherwise, $X_i \leftarrow 0$ and $y_i \leftarrow 0$. Therefore, the new position of $c$ is $(\sum_{i=1}^{n} X_i)/(\sum_{i=1}^{n} y_i)$. If $\sum_{i=1}^{n} y_i \neq 0$. In Step 2 and Step 3 of the Algorithm 3-1, we will use the vectors summing to zero scheme to avoid the easy guessing of point-to-cluster assignments.

Algorithm 3-1: Compute the new center position for cluster $c$.

1. $P_1$ creates an encryption-decryption key pair $(e, d)$ of one of the homomorphic encryption schemes and broadcasts $e$ to other parties.
2. $P_1$ arbitrarily creates $n \times 1$ random vectors $V_1, V_2, \ldots, V_n$ and $n$ random numbers $a_1, a_2, \ldots, a_n$ such that $\sum_{i=1}^{n} V_i = 0$ and $\sum_{i=1}^{n} a_i = 0$ for $i = 1, \ldots, n$, and then sends $V_i$ and $a_i$ to $P_1$.
3. $P_i, i = 1, \ldots, n$, sends $e(X_i + V_i)$ and $e(y_i + a_i)$ to $P_2$.
4. $P_2$ calculates $e(\sum_{i=1}^{n} X_i) = \prod_{i=1}^{n} e(X_i + V_i)$ and $e(\sum_{i=1}^{n} y_i) = \prod_{i=1}^{n} e(y_i + a_i)$, for $i = 1, \ldots, n$.
5. $P_2$ arbitrarily chooses $\alpha \in \mathbb{N}$, and calculates $e(\sum_{i=1}^{n} X_i) = e(\sum_{i=1}^{n} X_i)^\alpha$ and $e(\sum_{i=1}^{n} y_i) = e(\sum_{i=1}^{n} y_i)^\alpha$, for $i = 1, \ldots, n$.
6. $P_2$ sends $e(\sum_{i=1}^{n} X_i)$ and $e(\sum_{i=1}^{n} y_i)$ to $P_1$.
7. If $\alpha \sum_{i=1}^{n} y_i = 0$, $P_1$ arbitrarily chooses a new position and broadcasts to other parties. Otherwise, $P_1$ calculates $(e(\sum_{i=1}^{n} X_i))/(e(\sum_{i=1}^{n} y_i))$ and broadcasts to other parties.

We discuss the correctness, privacy issues and time complexities of Algorithm 3-1 below.

It is not difficult to see that the above algorithm works correctly. As for the privacy issues we consider the following. When a party sends its data of a point to $P_2$, its data is mixed with a random vector only known to $P_1$ and encrypted by an encryption key that can only be decrypted by $P_1$. When $P_1$ receives the data, it is already multiplied by a factor $a$, which is only known to $P_2$. Thus the first privacy issue is addressed. The second privacy issue is addressed with vectors summing to zero. With that $P_2$ can only get data mixed with random vectors. So the relations of points and centers will not be revealed. For the last privacy issue, $P_2$ first gets $e(y_i + a_i)$ in Step 3, and the privacy is protected by random number $a_i$ and homomorphic encryption. Later $P_2$ gets $e(\sum_{i=1}^{n} y_i)$ in Step 4, which is semantically secure by homomorphic encryption. In the final step, $P_1$ gets $a \sum_{i=1}^{n} y_i$, which is protected by the random number $a$.

As for computational complexity, Step 1 takes $O(1)$ time, Step 2 takes $O(nm)$ time, Step 4 takes $O(nm)$ time, and Step 5 and 7 both take $O(m)$ time. For communication complexity, Step 1 takes $O(1)$ time, Step 2 takes $O(nm)$ time, Step 3 takes $O(nm)$ time, and Steps 6 and 7 take $O(m)$ time. Since we need to compute $k$ new centers, it totally takes $O(nmk)$ time in both computational and communication complexity.

In Step 2 of Algorithm 3-1, $P_1$ needs to distribute $n \times 1 \times m$ random vectors and $n$ random numbers to $P_1, P_2, \ldots, P_n$, and it takes $O(nm)$ time for each center. As a first step to speed it up, we consider a height-balanced tree $T$ of bounded degree, where (1) the vertex set is $P_1, P_2, \ldots, P_n$, (2) the edges form a height-balanced tree with $P_1$ as the root, and (3) the height of
Now in each group, the \( k \) parties have collected the data of the \( k \) centers. For a center \( c \), its data is collected into one of the parties in each group. If we collect all the data of these \( O(n/k) \) parties, the new position of \( c \) can be calculated. We can use the \( O(m \log n) \) method presented above, and the time complexity is \( O(m \log(n/k)) \) for center \( c \) since the number of parties is \( O(n/k) \). Since the parties possessing the data of the \( k \) centers are disjoint, they can process in parallel simultaneously.

Phase 1 needs \( O(mk) \) time and phase 2 needs \( O(m \log(n/k)) \) time, and the totally time complexity is \( O(mk + \log(n/k)) \). This algorithm is for solving the bottleneck steps of Algorithm 3-1, Step 2 to 4. Thus the totally time complexity for solving this problem is reduced to \( O(mk + \log(n/k)) \).

**Theorem 3.2:** Privacy-preserving Multi-party k-means Clustering Problem for horizontal partitioned data can be solved in time complexity \( O(mk + \log(n/k)) \). \( \Box \)

**More discussions:** If we use the spirit of Divide-and-Conquer (D&C) to solve this problem, the recurrence relation will be like below:

\[
T(n) = T(n/2) + 1 \quad n > k
\]

\[
T(n) = k \quad n \leq k
\]

The solution of this recurrence relation is \( O(k + \log(n/k)) \).

Timing the message size \( O(m) \) becomes \( O(m(k + \log(n/k))) \). Actually, our method presented above can be regarded as a bottom-up implementation of the D&C algorithm.

**IV. CONCLUSION**

In this paper, we have introduced parallel computing into privacy-preserving multi-party \( k \)-means clustering problem for a set of \( n \) points in \( \mathbb{R}^m \), under two data partition models: vertical and horizontal partition model. This is the first paper introducing parallelism to this problem. The time complexities for vertically and horizontally partitioned data are \( O(nk) \) and \( O(mk + \log(n/k)) \) respectively, in both computational and communication complexity.\(^6\) Each of the data partition model has associated with it a certain hot spots, e.g., assignment of points to clusters and computation of new centers, when addressing the privacy issues. When the data distribution is arbitrary, this problem may be more complicated and worth further investigation. Whether the idea of parallel computing can be further extended and applied to other problems with privacy consideration remains to be seen. It is our hope that more privacy-preserving data mining problems can be identified for which the idea of parallelism can be found fruitful.

**V. ACKNOWLEDGEMENT**

This work was supported in part by the National Science Council, Taiwan, under the Grants NSC 98-2221-E-001-007-MY3 and NSC 98-2221-E-001-008-MY3, and by the Taiwan

\(^6\)For communication complexity, the complexity depends on the size of messages, or package of data, which is assumed to be a constant.
Information Security Center (TWISC) under the Grants NSC 97-2219-E-001-001 and NSC 97-2745-P-001-001.

Part of this work was completed during the visit of Teng-Kai Yu at the Carnegie Mellon CyLab Japan, Kobe, Japan.

REFERENCES


ceedings of the 14th ACM conference on Computer and communications


2003.

[9] I. Dhillon, D. Modha. A Data-Clustering Algorithm On Distributed
Memory Multiprocessors. In Large-Scale Parallel Data Mining, Lecture
Notes in Artificial Intelligence, 2000.

large graphs via the singular value decomposition. Machine Learning,


[12] K. Fukunaga. Introduction to Statistical Pattern Recognition. Acade-

product computation for privacy-preserving data mining. In The 7th
Annual International Conf. in Information Security and Cryptology,
2004.


diagrams and randomization to variance-based clustering. In Proc.

Privacy preserving clustering on horizontally partitioned data. Data and

[21] G. Jagannathan, R. N. Wright. Privacy-Preserving Distributed k-
Means Clustering over Arbitrarily Partitioned Data. In Proceedings of the
eleventh ACM SIGKDD international conference on Knowledge

[22] S. Jha, L. Kruger and P. McDaniel. Privacy Preserving Clustering,
10th European Symp. on Research in Computer Security, pages 397-
417, 2005.

[23] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and
A. Wu. A local search approximation algorithm for k-means clustering.

approximation algorithm for k-means clustering in any dimensions, in
2004.


[26] S. Lloyd. Least squares quantization in PCM. IEEE Transactions on

[27] M. Mahajan, P. Nimbhorkar and K. Varadarajan. The Planar k-means
Problem is NP-hard. In Proceedings of the 3rd International Workshop
on Algorithms and Computation, Kolkata, India, LNCS Vol. 5431, pages

[28] R. Ostrovsky, Y. Rabani, L. Schultman, and C. Swamy. The effective-

[29] P. Paillier. Public-key cryptosystems based on composite degree resid-
osity classes. In: Advances in cryptography-EUROCRYPT ’99, Prague,

in Vertically Partitioned Data. In Proceedings of The 8th ACM SIGKDD
International Conference on Knowledge Discovery and Data Mining,

[31] J. Vaidya and C. Clifton. Privacy-Preserving k-Means Clustering over
Vertically Partitioned Data. Proc. 9th ACM SIGDD Intc. Conf. on


University of Ottawa, Canada, 2006.

[36] J. Zhan, S. Matwin, L.W. Chang. Privacy-preserving collaborative asso-
ociation rule mining. Journal of Network and Computer Applications,