Parallel Differentially Private K-Means Implementation Using COMPSs Framework

Sukgamon Sukpisit*, Srdjan Skrbic*, Dusan Jakovetic*

*University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Novi Sad, Serbia
sukgamon.s@psu.ac.th, srdjan.skrbic@dmi.uns.ac.rs, dusan.jakovetic@dmi.uns.ac.rs

***Abstract*— K-means is one of the most important clustering algorithms, but it does introduce a risk of privacy disclosure in the clustering process. One approach to solving this problem is by applying differential privacy to K-means clustering algorithm to effectively prevent privacy disclosure. Increasing amounts of information generated in big data processing scenarios make clustering a challenging task. In order to deal with the problem, various approaches to the parallelization of clustering algorithms have been attempted. This paper presents an implementation of a differentially private k-means clustering algorithm that uses ε-differential privacy, based on the COMPSs framework for parallel computing. The experimental results show that the proposed implementation scales well and can be used to efficiently process large datasets using high-performance computing equipment.**

I. INTRODUCTION

With a variety of proposed techniques, machine learning can perform a classification, clustering or even prediction with a reasonable accuracy depending on the data it has been trained with. Therefore, data feeding to a machine learning algorithm should be accurate and well prepared. Training data could contain sensitive fields such as a living town or income. Combining these kinds of data, an adversary who knows some background or parts of the data could identify a participant in the dataset or extract sensitive data. As an attempt to solve this problem, Dwork proposed differential privacy algorithm that introduces random noise to protect sensitive data [1].

The accuracy of the model also depends on the amount of data that have been used as a training dataset. With the rapid growth of data in size, training processes consume more time. To reduce the time for training a model, we use high-performance computing framework that automates parallelization of code called COMPSs [2]. The framework distributes specified tasks and portions of data to available nodes and aggregates results.

In 2018, Zhang et al. proposed a parallel differentially private K-means clustering algorithm in [3]. The proposed algorithm uses the contour coefficients to evaluate each clustering iteration and add noise to centroids.

We propose the implementation of parallel differential privacy K-means clustering using COMPSs framework. We test the implementation using high-performance computing equipment. The results show that the implementation scales well and is usable in big data processing scenarios that rely on application of high-performance computing.

II. DIFFERENTIAL PRIVACY K-MEANS CLUSTERING AND COMPSS FRAMEWORK

A. *Differentially private K-means*

Dwork proposed applying differential privacy on K-means clustering which satisfies ϵ-differential privacy in [4]. The noise is added when the centroid in each K cluster is updated. For $1 \leq j \leq K$, the updating of a new centroid $\acute{c}_j$ in a cluster S_j, where vector p_i is the i-th member vector of a cluster S_j can be expressed as follows:

$$\acute{c}_j = \frac{\sum_{i \in S_j} p_i}{|S_j|} \qquad (1)$$

To calculate noises that will be added to both numerator and denominator of equation (1), the sensitivity values must be determined. For the denominator, an absence of the *x-th* vector which is a member of the cluster causes the number of members in the cluster changes by 1. Hence, the noise added to the denominator is $Lap(\frac{1}{\epsilon})$ regarding the Laplacian mechanism.

The sensitivity value for the numerator is determined by the dimension *dim* of a vector. Adding or removing a vector from a cluster can affect the sum by at most 1 for each dimension. Thus, the sensitivity for a sequence query is $dim + 1$. The updating a new centroid applying differential privacy is changed to equation (2). Where the scale parameter *b* in the numerator is varying depending on the following cases:

a) If the number *N* of iterations is specified, then $b = \frac{(dim+1)N}{\epsilon}$.
b) In the case of *N* is unknown, the parameter *b* can be obtained by increasing the noise parameter repeatedly for each iteration. For example, the parameter *b* in the first iteration can be $\frac{(dim+1)}{(\epsilon/2)}$ and changed to $\frac{(dim+1)}{(\epsilon/4)}$ in the next iteration.

$$\acute{c}_j = \frac{\sum_{i \in S_j} p_i + Lap(b)}{|S_j| + Lap(\frac{1}{\epsilon})} \qquad (2)$$

B. *COMPSs Framework*

COMPSs is a programming framework that supports parallel computing with C/C++, Java, and, Python programming language in distributed environments. COMPSs provides a programming model based on sequential programming development and execution of applications in cloud infrastructures. This framework implements a task-based programming model. Developers write a sequential program and specify the parts they want to be parallelized.

Since the COMPSs runtime has been implemented in Java, running a Java application on COMPSs does not require an additional API. However, PyCOMPSs package for development in Python has also been provided.

III. The Implementation

In K-Means clustering, vectors in dataset *D* are grouped into *K* clusters. The *K* centroids (abbreviated as *C*) are initiated by randomly selected *K* vectors from *D*. To enable parallelization, *D* is divided into *n* smaller subsets S_0 to S_{n-1}. It is important that the subsets have as equal size as possible because of the balancing of computational time of tasks being processed in parallel.

The next step is to perform the first clustering iteration. Since COMPSs employs task-based parallelism, we will explain the processes in the form of tasks. Fig. 1 shows the task dependency graph of the clustering process.

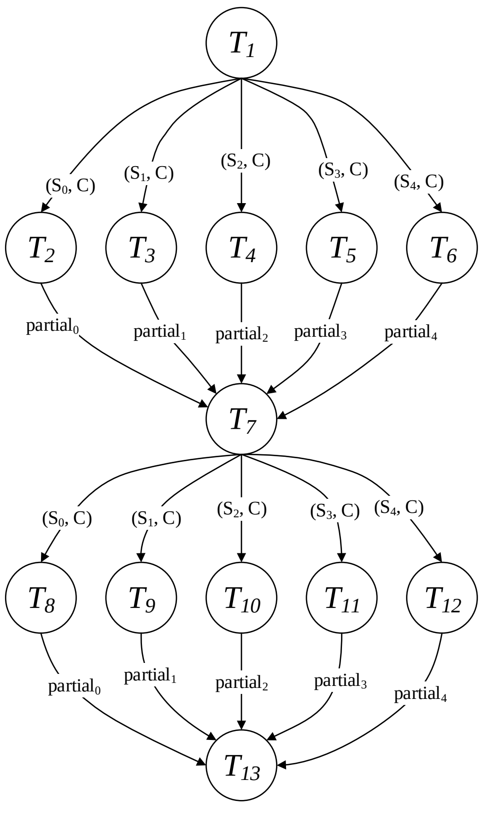

Fig. 1 Tasks dependency graph of the clustering process

Suppose the initialization process mentioned above is handled by T_1 (denotes for Task No. 1). The *n* subsets of *D* are distributed to the other n tasks. For example, if *D* contains 1,500 tuples, and *n* = 5, *D* is divided into 5 subsets, each holds 300 tuples. T_2 handles S_0, T_3 handles S_1, and so on. Each task finds the nearest centroid for vectors it handles, performs sum of dimensions and calculates the size of each cluster. At this stage, the newly created task T_7 synchronously collects information from T_2 to T_6, combines it and re-calculates new centroids $\acute{C}$. The clustering process is performed iteratively until C and $\acute{C}$ converge or the configured maximum of iterations is reached.

The Laplace noise is applied to the updating centroids step to make our implementation satisfy ϵ-differential privacy. The random selected noise will be added in both the numerator and denominator with different base parameters. For the numerator, we gradually increase the privacy parameter in each iteration (option b), and, $\frac{1}{\epsilon}$ for the denominator. Fig. 2 contains an outline of our implementation of the parallel differential privacy K-means clustering algorithm. The annotation @Parallel decorates the part that is processed in parallel.

IV. Experimental Results

For the purpose of testing, we randomly created a dataset of real numbers containing five million tuples with 8 dimensions. This dataset was tested on COMPSs cluster infrastructure consisting of 6-core Core i7 CPU nodes interconnected by the 10Gbps Ethernet network. Fig. 3, 4 and 5 show trends of the execution times, speedups, and efficiencies related to the number of processes used in computation.

```
Algorithm: Parallel Differential Privacy K-Means Clustering
Inputs: number of clusters K, privacy budget epsilon, number of processor n, dataset D, dimension dim
Output: centroids $\acute{C}$
Start
Divide D into n subsets $S_0, \ldots, S_{n-1} \subset D$
C ← initial K centroids
Add noise to C
$\acute{C}$ ← NULL
While C and $\acute{C}$ are not converged
  If $\acute{C}$ is not NULL then
    C ← $\acute{C}$
  End if
  @Parallel: Start
  For every i from 0 to n-1:
    partial_sum += Find_nearest_centroid($S_i$, C)
  End For
  @Parallel: End
  $\acute{C}$ ← Recalculate_centroid(patial_sum, dim)
End While
Return $\acute{C}$
```

Fig. 2 Outline of the algorithm implementation.

The dataset is divided into subsets equal to the number of processes to maximize the efficiency of the parallel computation. Therefore, adapting the size of subsets according to the number of processes used in the computation was necessary.

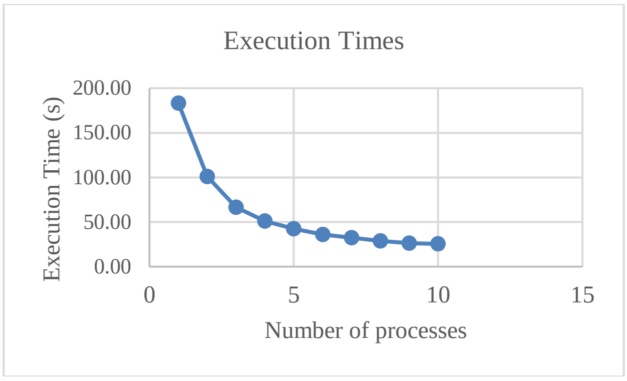

Fig. 3 Trends of execution times.

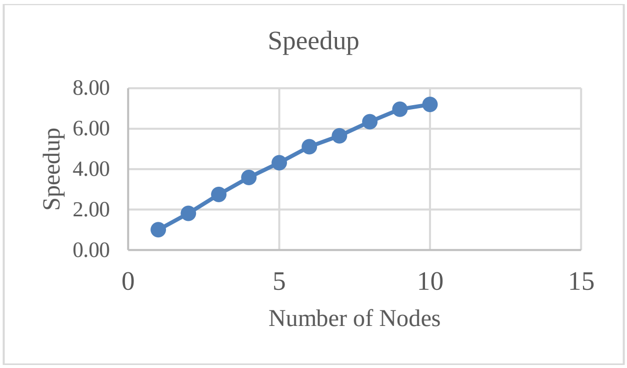

Fig. 4 Measured speedup.

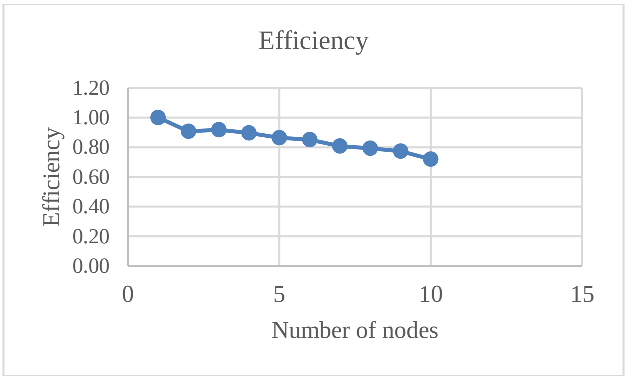

Fig. 5 Measured efficiency trends.

For measuring the impact of the privacy budget on the output, we used 3 well-known datasets from UCI Machine Learning Repository: Iris, Wine, and Yeast.

The Silhouette scores for those datasets are obtained by the parallel KMeans algorithm provided in dislib library. The Silhouette score is used for determining how well data in the dataset are clustered [5] ranged from -1 to 1. The high values near 1 indicate that the molecule is far from the other clusters while the values near -1 indicate that the molecule might be assigned to the wrong cluster. Table I shows the Silhouette score for each dataset. The abbreviation NCM refers to the number of clusters having members.

There are important issues in this experiment. First, with the lower privacy budget, e.g., epsilon = 0.1, the large noises will be added in the re-calculating centroids process. These noises stretch the centroids to be far from each other and cause all samples to be assigned to the single cluster. In this case, the Silhouette score cannot be calculated.

Another issue is the number of clusters having members does not match with the number of clusters. Suppose we used K=3, with low possibility, we might get two clusters containing members plus one empty cluster. This means the empty cluster is positioned too far from the rest. In this case, the Silhouette score could be high and difficult to compare with the normal KMeans algorithm which outputs 3 clusters containing members.

TABLE I.
THE DATASETS AND ITS SILHOUETTE SCORE

Dataset	Samples	Features	K	Silhouette Scores	NCM
Iris	150	4	3	0.5510	3
Wine	178	13	3	0.5711	3
Yeast	1484	8	10	0.1695	10

To avoid the problem mentioned above, we first find the least epsilon value that produces K members contained clusters for each dataset. Then, we slightly adjust the epsilon value to observe the change of the Silhouette score.

It is important to know that a dimension of a dataset directly affects the magnitude of noises added. It makes the least epsilon value for outputting K members contained clusters for one dataset is different from the others. Table II. shows the impact of the privacy budget on the Silhouette score.

TABLE II.
THE IMPACT OF THE PRIVACY BUDGET ON THE SILHOUETTE SCORE

Iris		Wine		Yeast	
Epsilon Value	Silhouette Score	Epsilon Value	Silhouette Score	Epsilon Value	Silhouette Score
2.9	0.5067	**0.4**	0.5544	**86.2**	0.1607
3.5	0.5284	**1**	0.5611	**94.7**	0.1630
4	0.5350	**1.5**	0.5677	**95**	0.1678

From the result shown in Table II, we can see the greater privacy budget yields the better Silhouette score and closer to the result of the normal parallel KMeans algorithm.

V. CONCLUSION

We implemented differentially private K-means clustering algorithm using high-performance computing framework - COMPSs. To achieve ϵ-differential privacy, we employed the approach proposed by Dwork in which the selected random noise was added to updating centroids step using Laplacian mechanism. We performed experiments with our implementation using distributed environment. The five million tuples dataset was randomly created and divided into smaller subsets. The number of subsets related to the number of processes available in our computer cluster. The dataset was tested multiple times with various numbers of processes used. The experimental results show that our implementation scales well and achieves speedups not lower than 70% of linear speedup, for up to 10 processes.

For measuring the privacy-preserving of our algorithm, we calculated the Silhouette score for epsilon values varying by datasets comparing to the score obtained by the normal parallel KMeans algorithm. The results indicate that the greater privacy budget yields more accurate output.

Acknowledgment

This work is supported by the I-BiDaaS project, funded by the European Commission under Grant Agreement No. 780787. This publication reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein. The authors gratefully acknowledge the AXIOM HPC facility at the Faculty of Sciences, University of Novi Sad, where all the tests were run.

References

[1] C. Dwork, "Differential privacy" *Automata, Languages and Programming, 33rd International Colloquium, ICALP2006, Venice, Italy, Proceedings, Part II*, pp. 1–12, 2006.

[2] R. M. Badia, J. Conejero, C. Diaz, J. Ejarque, D. Lezzi, F. Lordan, C. Ramon-Cortes, R. Sirvent, "COMP Superscalar, an interoperable programming framework", *SoftwareX*, vol. 3–4, pp. 32-36, 2015.

[3] Y. Zhang, N. Liu, and S. Wang S, "A differential privacy protecting K-means clustering algorithm based on contour coefficients," *PLOS ONE,* vol. 13(11), e0206832, 2018.

[4] C. Dwork, "A firm foundation for private data analysis," *Communications of the ACM*, vol. 54, pp. 86-95, 2011.

[5] P. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," Computational and Applied Mathematics, vol. 20, pp. 53-65, 1987.