Parallel K-Means Clustering for Gene Expression Data on SNOW

Briti Deb
Institute of Computer Science, University of Tartu,
J.Liivi 2, Tartu, Estonia

Satish Narayana Srirama
Institute of Computer Science, University of Tartu,
J.Liivi 2, Tartu, Estonia

ABSTRACT
The exponential growth in the amount of data brings in new challenges for data analysis. Gene expression dataset is one such type of data necessitating analytical methods to mine patterns implicit in it. Although clustering has been a popular way to analyze such dataset, the increase in size of dataset necessitates the need for improving the efficiency of clustering methods. In this paper, we study the use of using Principal Components (PCs) as a pre-processing step to provide a more efficient data structure to a parallel formulation of the sequential K-Means algorithm, utilizing multiple cores available in a desktop computer, via the Simple Network of Workstations (SNOW) package. Initial result suggests that SNOW package provides an intuitive way for biologists to parallelize algorithms and speedup job execution, particularly for jobs like K-Means clustering which depends on random starting centroid locations.

General Terms
Parallel K-Means

Keywords
SNOW, Parallel K-Means Clustering, Scalability Testing

1. INTRODUCTION
The exponential growth in the amount of data, popularly known as big data, brings in new challenges for data capture, storage, search, transfer, curation, visualization, accessibility, and analysis. Big data analysis is widely believed to be one of the next frontiers for innovation, scientific discovery, business competition, and industrial productivity. Gene expression dataset is one such type of big data, necessitating analytical methods to mine patterns towards understanding the functions and structural organization of genes. One of the popular ways to analyze gene expression dataset is clustering, such as (sequential) K-Means. With the increase in the size of data, the need for improving the efficiency of clustering methods grows. Two main directions have been explored to improve the efficiency of sequential K-Means. First, using various parallel formulation of the sequential K-Means algorithm which leverage multiple processing units [1][2][3], and second, by adopting a more efficient data structure/algorithm [4]. In this paper, we study the use of using Principal Components (PCs) as a pre-processing step to provide a more efficient data structure to the parallel formulation of the sequential K-Means algorithm, utilizing multiple cores available in a desktop computer, via the Simple Network of Workstations (SNOW) package [5]. We employed a variant of the popular parallel programming style based on message passing model, known as the Single Program Multiple Data (SPMD) [6], where each worker have their own replica of the data, and multiple processing units simultaneously execute the same program with different inputs to obtain results faster. Initial result suggests that it is convenient to adopt this technique for small and medium parallel environments.

In this work, we study the combined pre-processing by PCA and parallel formulation for the sequential K-Means algorithm using multiple cores available in a desktop computer using the SNOW package. There are several contributions of our approach. First, we implemented the parallel K-Means algorithm for PCA pre-processed data leveraging multiple cores available in a desktop computer. Second, we experimentally analyzed the load balancing and speedup issues of the parallel algorithm. Third, we validated the obtained clusters by statistical validation.

The rest of the paper is organised as follows. Section 2 discusses related works in Principal Component Analysis (PCA) and parallel K-Means clustering, followed by our proposed algorithm in section 3. Section 4 provides an experimental analysis of the proposed algorithm. Finally, section 5 provides conclusive remarks, limitations of our approach, and future research directions.

2. LITERATURE SURVEY
Clustering data to discover interesting patterns is an important process within the field of data mining. In a review by [7], three broad classes of clustering algorithms has been identified. The first is the heuristics-based algorithms such as K-Means whose output depends on starting centroid locations, second is the model-based algorithms such as mixture models, and third is the density-based algorithms such as DBSCAN. Among the widely used clustering algorithms are the K-Means, hierarchical, and model-based clustering, each having its own pros and cons. K-Means require pre-defining the number of clusters (k) and is dependent upon the random starting centroid locations, whereas hierarchical clustering does not produce hard-clustering, leaving a challenge to the user to interpret the resulting dendogram. The K-Means algorithm is also an NP-hard optimization problem [8], which makes it impractical to apply classical problem solving methods to find an exact solution. Given such challenges in K-Means, attempts have been made to obtain satisfactory solution by following greedy approach, also knows as heuristics. Following such an approach, the K-Means method partitions n observations into k clusters, where each observation belongs to the cluster with the nearest mean, and the convergence is determined by using a squared error distortion measure.

Due to the problem of immense growth in the amount of data, both in terms of size and dimensionality, the sequential K-Means algorithm could face several challenges such as load balancing and speedup. Several studies have been conducted to adapt the the sequential K-Means for a parallel version to improve the efficiency while handling big data [4]. Largely, two main directions have been explored in this area. First, distributing data and computation loads over multiple processing units, by using parallel computing frameworks such as MapRreduce [9], Iterative MapReduce, Bulk Synchronous Parallel, Graphics Processing Units, and Message Passing Interface (MPI) based frameworks such as SNOW [10]. Second, by adopting a more efficient data structure/algorithm, such as pre-processing the data with Principal Component Analysis (PCA) to reduce data dimensionality and the amount of computation [11][12]. However, little work has been done to provide a combined pre-processing with PCA and parallel formulation of K-Means. In this work, we study the combined pre-processing by PCA and parallel formulation for the K-Means algorithm leveraging multiple cores available in a desktop computer via the SNOW package.

PCA [12] is a data dimensionality reduction method by combining two or more (possibly) correlated variables into a new factor variable and variance maximizing rotation of the original variable space. It is a method commonly used to find patterns in high-dimensional data. For a square matrix A, given X as the eigenvector and λ as the eigenvalue, we can write $AX = \lambda X$. Once the eigenvectors are found, they are ordered according to decreasing eigenvalues. The highest eigenvalue indicates the most "significant" eigenvector having the highest variance of data. The new variables known as the Principal Components (PCs) are linear combinations of original variables. PCA uses co-variance analysis between the variables, to reduce the observed variables into a smaller number of principal components (PCs).

3. PROPOSED ALGORITHM

3.1 Data Preprocessing with PCA

The data was preprocessed using PCA, enabling to conduct experiments using different number of PCs. A primary challenge after PCA is to decide on how many components (or factors) to retain, as there exists no theoretical reason for choosing any particular number of PCs to retain. One popular method used to decide on the number of PCs to retain is the scree test [13], which in our case suggested to retain the first two PCs giving 18 % cumulative proportion of variance. We experimented using different number of PCs to analyze scalability and speedup issues. Loadings and scores are used to extract patterns from the PCs. The loadings (rotation matrix) which define the new coordinate system, indicate the size of the contribution of each original variables to the PCs. The scores are formed by multiplying the loadings with the original data. Scores are obtained to see the original data in terms of PCs, after projecting the data to the new coordinate system represented by the eigenvectors.

3.2 Initializing Master and Workers

We experiment a parallel approach using the SNOW [5] parallel programming package available for R, which enables us to use multiple cores available in a desktop computer, to distribute the tasks and execute the job faster. The parallelization of K-Means in SNOW has been achieved by adapting the sequential K-Means to leverage multiple cores available in a desktop computer. The R functions are executed in parallel using variations of the standard *lapply()* function in R. The parallel function execution in SNOW starts with creating a cluster object using *makeCluster()*, which is used to interact with the cluster workers. The cluster object is created by specifying configuration options such as the number of workers and the transport mechanism between the workers and the master, which can be socket or MPI. The data can be distributed to each worker using functions such as *clusterCall()*, *clusterExport()*, among others. The *clusterCall()* method takes as arguments a SNOW cluster object, function to be executed in the workers, and arguments to pass to the function, which is then called on each of the workers. We distributed the data to workers using the *clusterExport()* function, and used MPI as the transport mechanism between the workers and the master.

3.3 Parallel K-Means

The aim of this paper is to study the load balancing and speedup of parallel K-Means via SNOW under different dimensions of data and different number of computing nodes. We implement the parallel K-Means by adapting the sequential K-Means for data pre-processed by PCA, leveraging multiple cores (hereafter also referred to as workers or nodes) available in a desktop computer. This approach is similar to the popular parallel programming style SPMD, where multiple workers simultaneously execute the same task with different inputs to obtain results faster. As the result of K-Means (and many other machine learning algorithms such as bootstrapping, cross validation, among others) depends upon the random starting centroid locations, it becomes necessary to experiment with several random starting points [5]. In the sequential K-Means, this is done by executing all the iterations in a single compute node, and as the number of iterations grow, the execution time gets slower. In our approach, the random starting points are distributed to several workers, aiming to gain speedup. The parallel K-Means is based on a master/worker architecture [5], where the master send tasks to the workers, who execute the tasks, and returns the results to the master, which aggregates the results, and select the cluster with the smallest within-cluster sum of squares (WCSS) as the solution to the clustering problem.

The SNOW parallel programming package enables us to use multiple cores available in a desktop computer, to distribute the tasks and execute the job faster. The workers generate different random starting numbers (*nstart*), and to prevent them replicate each others results, SNOW provides two ways to generate the random numbers [5]. First, seeding the workers differently in an ad-hoc scheme, and second, using parallel random number generation packages such as *rspring* and *rlecuyer*.

The *clusterApply* function is used to parallelize the algorithm, which takes as parameters the division of *nstart* workload among workers, and the *kmeans()* parameters. The *kmeans()* parameters consists of the number of cluster centers *k*, and the *nstart* value. The algorithm randomely select *k* rows from the dataset as random starting centroid locations, assigns the *n* observations to the *k* clusters, such that each observation belongs to the cluster with the nearest mean, recalculate the centroid of each of the *k* clusters as the new mean, until convergence has been reached. This *nstart* argument in *kmeans()* specifies the number of times clustering solutions would be obtained using different random cluster centers. After *nstart* number of cluster solutions have been obtained

in the workers, the results are sent to the master, which combines the results and compute the smallest WCSS which is taken as the final solution of the K-Means clustering problem. The pseudocode of the parallel K-Means is shown in Algorithm 1.

Algorithm 1. Parallel K-Means

Input: set of data points *n* to be clusered,
number of clusters *k*,
number of maximum iterations *maxIters*,
task vector {*nstart* , *w*} where *nstart* is number of times random centroids computed in each worker,
number of available workers *w*,
Output: set of *c* cluster centroids, set of cluster labels of *n*,
Steps:
PCA Data Pre-processing Steps:

1. Input data *n*
2. Normalize data by mean centering
3. Compute covariance matrix
4. Compute the eigenvectors and eigenvalues of the covariance matrix
5. Choose components and form the feature vector
6. Derive the new data set *n1*

Parallel K-Means Clustering Steps:

7. Spawn *w* workers
8. for each $i \in w$ do
9. for each $j \in nstart$
10. Decide the number of clusters *k* and the random starting centroid locations in *n1*
11. Assign each data point to the cluster represented by the mean it is nearest to
12. Move each mean to the actual mean of the data points in its cluster
13. Stop when *maxIters* reached or the means stop moving, otherwise go back to 12
14. Compute WCSS and append centroids and WCSS in a variable
15. end for
16. end for
17. Workers send results to the master
18. Master aggregates the results and selects the cluster with the smallest WCSS as the K-Means solution

4. EXPERIMENTAL ANALYSIS

Experiments have been performed using the SNOW package for R. The experimental tests were carried out on an Intel Core-i5 4-core processor computer with 4 GB of memory running on Ubuntu Linux.

We began our experiments with the parallel K-Means algorithm using the *clusterApply()* function, which schedule the tasks in a round-robin fashion, or in other words, the master pushes the tasks to the workers [5]. Instead of using a single worker to execute all the *nstart* values, the parallel K-Means algorithm distributes the *nstart* to different workers to obtain faster results. We used a smaller value of the *nstart* argument (for instance, a vector of four 2500s) in the *kmeans()* function call on each of the workers (for instance, four workers), which is equivalent to using *nstart* = 10000 in a single call to the sequential *kmeans*. The workers send all their cluster solutions to the master, which combine the results and select the result with the smallest WCSS as the solution to the clustering problem. A timer has been used to record the running time of the whole process.

Next, we consider different situations, such as some workers may be slower than others, or some tasks may take longer time than others, or length of the (task) vector could be greater than the total number of worker nodes, in which cases the scheduling technique used becomes critical for performance. In this context, we experimented the utility a specific load balancing approach via the *clusterApplyLB()* function, which is aimed at reducing the time wasted due to overheads in round-robin scheduling. Instead of the master pushing the tasks to the workers as it is done in round-robin scheduling, the *clusterApplyLB()* lets the workers pull tasks from master as needed. The performance of load balancing is shown in Fig. 2, where we use *Sys.sleep()* function to compute the task lengths in each worker, and the *snow.time()* function to gather timing information about the overall execution. The performances of the parallel K-Means is evaluated on a gene expression dataset [14]. The computational speed of the parallel K-Means with load balancing as compared to sequential K-Means is shown in Table 1. The comparison of running time at different dimensions of data and different number of workers is shown graphically in Fig. 1(a-c), and comparison of speedup at different dimensions of data and different number of workers is shown in Fig. 1(d-f).

In Fig. 1(a-c), it can be seen that when the number of workers are increased from one to two to three, the execution time decreases. In Fig. 1(d-f) it can be seen that speedup increases when the number of workers are increased from one to two to three. Surprisingly, when data dimension is two PCs (Fig. 1-a), an increase in the number of workers from three to four results to an increase in execution time, which could be due to master-worker communication overhead.

Table 1: The execution time of serial K-Means vs parallel K-Means

Data Size (No of PCs)	Sequential KMeans Time (T_S) (sec)	No of Cores (C)	Load Balancing Parallel KMeans Time (T_C) (sec)	Speedup (T_S / T_C)
2	2.16	1	2.25	0.96
		2	1.53	1.41
		3	1.2	1.8
		4	1.34	1.61
200	75	1	97.1	0.77
		2	66.1	1.13
		3	56.2	1.33
		4	47.3	1.59
810	855	1	1024.2	0.83
		2	612.2	1.4
		3	450	1.9
		4	402	2.13

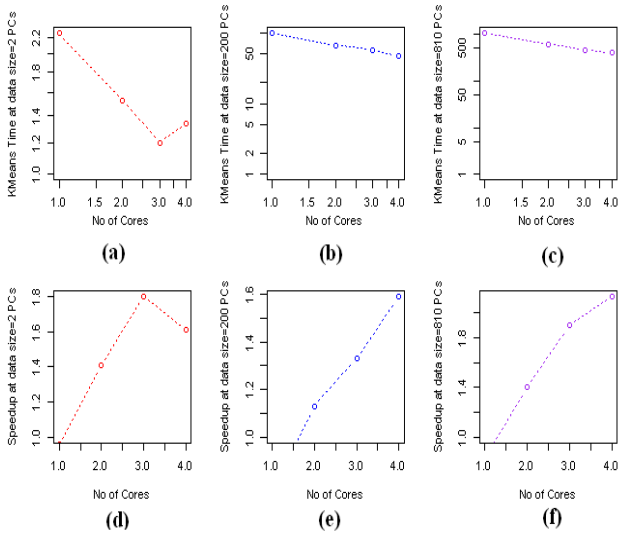

Figure 1: (a-c) Running time of parallel K-Means at different data dimensions and different number of workers, (d-f) Speedup of parallel K-Means at different data dimensions and different number of workers

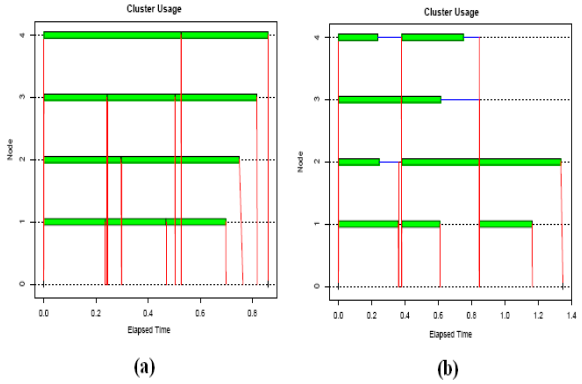

Figure 2: (a) Performance with load balancing for ten tasks in four cores, (b) performance without load balancing for ten tasks in four cores

Visible in Fig. 2 is the performance with and without load balancing, where solid horizontal bars indicate the time workers (nodes in Fig. 2) are busy. Breaks in the solid horizontal bars indicate time wasted due to scheduling. In the experiment, the job was divided into ten tasks, which were solved by four workers. Each horizontal bar segment indicates completion of one of the ten tasks. Fig. 2-b shows the utilization of workers without load balancing, where gaps in the horizontal lines indicates idle time of workers due to round-robin scheduling, resulting to more total time (1.3 sec) in job execution. Fig. 2-a shows the utilization of workers with load balancing, where the idle time of workers are much less, resulting to faster job execution (0.8 sec). This indicates that load balancing is important in better utilization of available workers when the length of the task vector is greater than the total number of workers. However, it must be noted that communication between master and workers can reduce performance. Furthermore, the allocation of nodes by scheduler is done in a nondeterministic fashion, which complicates the reproducibility in simulations.

To validate the obtained clusters, we used statistical validation using Silhouette Index (SI) [15]. Silhouette index is used to compare the similarity of various cluster solutions. A popular way to interpret the SI is as follows: for a well clustered observations the SI value is almost one, the SI values for observations which lies between two clusters is around zero, and the SI values for observations placed in the wrong cluster are negative. Using the first two PCs, we obtained an SI value of 0.47, indicating that some of the observations might lie between two clusters, which could be due to the use of only two PCs containing only 18% cumulative variance.

5. CONCLUSIONS AND FUTURE DIRECTIONS

Several conclusions can be made from this study. First, PCA can be used to reduce data dimensionality, and provide a more efficient data structure to the parallel formulation of the sequential K-Means algorithm. Second, the SNOW package could provide an intuitive way for biologists to parallelize algorithms which depends on random starting centroid locations such as K-Means, by utilizing multiple cores available in a desktop computer. Third, parallel K-Means with load balancing could speedup K-Means clustering jobs to some extent. Overall, we would recommend SNOW based parallel K-Means approach for clustering, leveraging multiple cores in a desktop computer to analyze large data.

Although currently getting popularity, SNOW has several drawbacks. First, it lack advance mechanisms to handle large distribution to workers, a challenge while working with big data. Second, the master keeps all the task results in its memory until they are returned to the caller, putting a burden on the memory.

In future, we would like to study the performance of parallel K-Means on SNOW using large clusters and large dataset. Also, we would like to perform more exhaustive cluster validation.

6. ACKNOWLEDGEMENTS

This research is supported by the European Regional Development Fund through the EXCS, Estonian Science Foundation grant ETF9287, Target Funding theme SF0180008s12 and European Social Fund for Doctoral Studies and Internationalisation Programme DoRa.

7. REFERENCES

[1] Dhillon, I. S. and Modha, D. S., A data-clustering algorithm on distributed memory multiprocessors. In Large Scale Parallel Data Mining, Lecture Notes in Computer Science, 1759:245–260, Mar. 2000.

[2] Judd, D., McKinley, P. K. and Jain, A. K., Large-scale parallel data clustering. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(8):871–876, Aug. 1998.

[3] Zhao, Weizhong, Huifang Ma, and Qing He. "Parallel k-means clustering based on mapreduce." Cloud Computing. Springer Berlin Heidelberg, 2009. 674-679.

[4] Pettinger, D. and Giuseppe D. F., "Scalability of efficient parallel K-Means." E-Science Workshops, 2009 5th IEEE International Conference on. IEEE, 2009.

[5] McCallum MC., Weston, S., Parallel R, Orielly publications, 2011

[6] Darema, F., SPMD model: past, present and future, Recent Advances in Parallel Virtual Machine and Message Passing Interface: 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, September 23–26, 2001. LNCS 2131, p. 1, 2001.

[7] Jain, A. K., and Richard C. D., Algorithms for clustering data. Prentice-Hall, Inc., 1988.

[8] Dasgupta, S., The hardness of k-means clustering. Department of Computer Science and Engineering, University of California, San Diego, 2008.

[9] Srirama, S. N., Batrashev, O. and Vainikko. E., "SciCloud: scientific computing on the cloud." Proc of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing. IEEE Comp. Soc., 2010.

[10] Tierney, L., et al. "Snow: simple network of workstations." R package version 0.3-3, URL http://CRAN. R-project. org/package= snow (2008).

[11] Johanson, K., et al. "Saccharomyces cerevisiae gene expression changes during rotating wall vessel suspension culture." Journal of Applied Physiology 93.6 (2002): 2171-2180.

[12] Ding, C. and Xiaofeng H., "K-means clustering via principal component analysis." Proceedings of the twenty-first international conference on Machine learning. ACM, 2004.

[13] Cattell, R. B., "The scree test for the number of factors." Multivariate behavioral research 1.2 (1966): 245-276.

[14] Adler P., et al, Mining for coexpression across hundreds of datasets using novel rank aggregation and visualization methods. Genome Biol. 2009;10:R139. Data: http://biit.cs.ut.ee/mem/training/

[15] Bolshakova, N., and Azuaje, F., "Improving expression data mining through cluster validation." Information Technology Applications in Biomedicine, 2003. 4th International IEEE EMBS Special Topic Conference on. IEEE, 2003.