Parallel Glowworm Swarm Optimization Clustering Algorithm based on MapReduce

Nailah Al-Madi, Ibrahim Aljarah and Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, ND, USA
{nailah.almadi,ibrahim.aljarah,simone.ludwig}@ndsu.edu

***Abstract*—Clustering large data is one of the recently challenging tasks that is used in many application areas such as social networking, bioinformatics and many others. Traditional clustering algorithms need to be modified to handle the increasing data sizes. In this paper, a scalable design and implementation of glowworm swarm optimization clustering (MRCGSO) using MapReduce is introduced to handle big data. The proposed algorithm uses glowworm swarm optimization to formulate the clustering algorithm. Glowworm swarm optimization is used to take advantage of its ability in solving multimodal problems, which in terms of clustering means finding multiple centroids. MRCGSO uses the MapReduce methodology for the parallelization since it provides fault tolerance, load balancing and data locality. The experimental results reveal that MRCGSO scales very well with increasing data set sizes and achieves a very close to linear speedup while maintaining the clustering quality.**

Keywords*—Big data clustering, Parallel Processing, Hadoop

I. Introduction

Data clustering [1] is one of the important data analysis tasks with the objective of dividing a set of unlabeled data objects into different groups called clusters; each group having common specifications between the group members. An efficient clustering algorithm is one that produces high quality clusters, in other words, the similarity measure between the data objects in the same cluster are to be maximized, and the similarity measure between the data objects from different groups are to be minimized. Clustering can therefore be formulated as a multi-objective optimization problem. Many techniques have been proposed to solve optimization problems such as swarm intelligence.

Swarm intelligence [2] simulates the natural swarms such as ant colonies, flocks of birds, bacterial growth, and schools of fish. The behavior of the swarm is based on the sense of the members' interactions in the swarm by exchanging their local information with each other to help reaching the food sources. There is no central member in the swarm, but rather all swarm members participate equally to achieve the goal. Some examples of swarm intelligence algorithms are Particle Swarm Optimization (PSO) [3], Glowworm Swarm Optimization (GSO) [4], and Ant Colony Optimization (ACO) [5].

In this paper, we use Glowworm Swarm Optimization (GSO) [4], which is inspired by simulated experiments of the behavior of insects called glowworms or lighting worms. These glowworms are able to control their light emission and use it for different objectives such as e.g., attracting other worms during the breeding season. GSO is especially useful for a simultaneous search of multiple solutions, having different or equal objective function values. A glowworm that emits more light (high luciferin level) means that it is closer to an actual desired position and has a high objective function value. A glowworm is attracted by other glowworms whose luciferin level is higher than its own within a local decision range. If the glowworm finds some neighbors with a higher luciferin level that are within its local range, the glowworm moves towards them. At the end of the process, most glowworms are gathered at the multiple peak locations in the search space.

Swarm intelligence has been effectively used for data clustering [6, 7]. Solving clustering as an optimization problem depends on the cluster quality measure (objective function) used such as cluster similarity, member intra-distance, sum of square errors, etc. Most swarm intelligence algorithms solve clustering as a global solution, where each individual searches for the complete clustering solution. However, GSO solves the optimization task in a different way, whereby each glowworm searches for one of the cluster centroids, which is considered as a sub-solution. The combination of these sub-solutions forms the global solution of the clustering problem.

Clustering very large data sets that contain large numbers of records with high dimensions is very difficult and computationally expensive. The parallelization of the algorithm is the solution to enable existing approaches to work feasibly on big data, which can be done using different methodologies such as MPI (Message Passing Interface) [8], or MapReduce [9], and many others. MapReduce is a prominent parallel data processing framework, which has been gaining significant interest from both industry and academia. It is a new methodology proposed by Google in 2004, which is a programming model and an associated implementation for processing large data sets [9]. MapReduce enables users to develop large-scale distributed applications by supporting fault tolerance, load balancing, and data locality.

In this paper, we propose a scalable design and implementation of glowworm swarm optimization clustering [10] using the MapReduce methodology called MRCGSO. MRCGSO is different from CGSO since it implements the Map and Reduce functions in order to achieve the goal of enabling CGSO to solve big data clustering and enhance its scalability. The proposed algorithm is evaluated using real and large synthetic datasets.

The organization of this paper is as follows: In Section II, we describe the glowworm clustering algorithm process and the MapReduce model, and introduce the proposed MRCGSO approach. Then, we report on our experiments, and show the results in Section III. We conclude this paper in Section IV.

II. Related Work

Big data clustering has recently received significant efforts to build parallel efficient and effective clustering algorithms. Most of these algorithms use the MapReduce methodology that has been devoted to enhance the algorithms' scalability.

Yaobin et al. in [11] introduced a MapReduce design and implementation of an efficient DBSCAN algorithm. The proposed algorithm tackled the drawbacks of existing parallel DBSCAN algorithms such as the data balancing, scalability limitation, and the limitation of the parallel environment used. The algorithm resolved these issues by using a fully parallelized implementation and by removing the sequential processing bottleneck as well as improving the algorithm's portability. Furthermore, the algorithm showed the biggest improvement when the data was imbalanced needing high computational cost. The evaluation experiments conducted using large scale datasets confirm the efficiency and scalability of their algorithm.

A parallel K-means algorithm clustering algorithm based on MapReduce was proposed in [12]. The algorithm finds the centroids by calculating the weighted average of each individual cluster points through the *Map* function; afterwards the *Reduce* function assigns a new centroid for each data point based on the distance calculations. Then, a MapReduce iterative refinement technique is applied to find the final centroids.

Ping et al. in [13] proposed an algorithm for solving the problem of the document clustering using the MapReduce-based K-means algorithm. The algorithm uses MapReduce to read the document collection to calculate the term frequency and inverse document frequency. The introduced algorithm represents the documents as [key, value] pairs, where the key is the document type and the value is the document text. The experiments showed that the algorithm works well for text clustering and can be done in relatively short time with high accuracy.

Li et al. in [14] suggested another K-means clustering algorithm using MapReduce by merging the K-means algorithm with ensemble learning method bagging. The algorithm is used to solve the outlier problem. Their algorithm shows that the algorithm is efficient on a large scale data sets with large number of outliers.

In [15], self-organizing map (SOM) was modified to work with large scale data sets by implementing the algorithm using Hadoop Mapreduce to improve the performance of clustering. Experiments were conducted with large real data sets and demonstrated the efficiency of MapReduce-based SOM and confirmed that the MapRduce methodology is applicable to parallelize these types of algorithms.

In [16], the authors used the MapReduce framework to solve co-clustering problems. They proved that MapReduce is a good solution for co-clustering mining tasks and they applied the algorithm to many application areas such as collaborative filtering, text mining, etc. The experiments showed that the co-clustering with MapReduce can scale well with large data sets.

In [17], a fast clustering algorithm with constant factor approximation guarantee was proposed. The authors use a sampling technique to decrease the data size. A comparison of this algorithm with several sequential and parallel algorithms for the k-median problem was done using randomly generated data sets to check the algorithm performance. The results showed that the algorithm achieves better or similar solutions compared to the existing algorithms especially on very large data sets.

Yang et al. in [18] proposed a big data clustering method based on the MapReduce framework. They used the ant colony approach to decompose the big data into several data partitions to be used in parallel clustering. Using MapReduce with the ant colony clustering algorithm lead to the automation of the semantic clustering to improve the data analysis. The proposed algorithm was developed and tested on data sets with large number of records and showed acceptable accuracy with good efficiency.

In [19], the authors suggested an idea to minimize the I/O cost for clustering analysis with the MapReduce model by minimizing the network overhead among the processing nodes. Furthermore, they suggested a subspace clustering method to handle very large datasets in efficient time. Experiments on data of millions of instances returned good speedup results.

All clustering algorithms applied on large scale data share the computational burden due to the high processing overhead and the iterative mechanism that is used for the implementation. To reduce the large computational burden, scalable algorithms can be achieved by employing parallel processing mechanisms to build parallel clustering algorithms. In this paper, a parallel clustering algorithm is proposed. We use glowworm swarm optimization as it performs multiple searches to find the best centroids in the search space. Furthermore, the MapReduce framework has been chosen as the parallelization technique in order to enhance the scalability and efficiency of the CGSO [10]. To the best of our knowledge, this is the first work that implements GSO clustering with MapReduce. The main objective for this paper is to show that GSO clustering benefits from MapReduce and is applicable for large datasets.

III. Proposed Approach

Given that our proposed approach is based on the glowworm swarm optimization clustering algorithm CGSO as well as the MapReduce methodology, we first briefly introduce GSO, then clustering using GSO and MapReduce before outlining the details of our proposed MRCGSO algorithm.

A. Glowworm Swarm Optimization Algorithm

GSO is one of the most recent swarm intelligence method introduced by Krishnan and Ghose in 2005 [4]. GSO is distinguished from other methods and has been efficiently used for optimizing multimodal functions. In GSO, glowworm swarm S, which consists of m glowworms, is distributed in the objective function search space. Each glowworm g_j ($j = 1...m$) is assigned a random position p_j inside the given search space. Glowworm g_j carries its own luciferin level L_j, and has a vision range called local-decision range rd_j. The luciferin level depends on the objective function value and the glowworm position. The glowworm with a better position is brighter than others, and therefore, has a higher luciferin level value and is therefore closer to the optimal solution. All glowworms seek the neighborhood set within their local decision range, and then move towards the brighter ones within the neighborhood set. Finally, most of the glowworms gather in compact groups in the function search space at multiple optimal locations.

GSO works in an iterative process that consists of several luciferin updates and glowworm movements, which are executed to find optimal solutions. The luciferin level L_j is updated using the following equation:

$$L_j(t) = (1 - \rho)L_j(t - 1) + \gamma F(p_j(t)) \tag{1}$$

where $L_j(t-1)$ is the previous luciferin level for glowworm j; ρ is the luciferin decay constant ($\rho \in (0, 1)$); γ is the luciferin enhancement fraction, and $F(p_j(t))$ represents the objective function value for glowworm j at current glowworm position (p_j); t is the current iteration.

After that, each glowworm j explores its own neighborhood region to extract the neighbors that have the highest luciferin level by applying the following rule:

$$z \in N_j(t) \;\; \textit{iff} \;\; Distance_{jz} < rd_j(t) \;\; and \;\; L_z(t) > L_j(t) \tag{2}$$

where z is one of the neighboring glowworms close to glowworm j, $N_j(t)$ is the neighborhood set, $Distance_{jz}$ is the Euclidean distance between glowworm j and glowworm z, $rd_j(t)$ is the local decision range for glowworm j, and $L_z(t)$ and $L_j(t)$ are the luciferin levels for glowworm z and j, respectively.

After that, each glowworm selects the movement direction using the roulette wheel method. Therefore, the glowworm position (p_j) is adjusted based on the selected neighbor position (p_z) using the following equation:

$$p_j(t) = p_j(t - 1) + s \, \frac{p_z(t) - p_j(t)}{Distance_{jz}} \tag{3}$$

$p_j(t - 1)$ is glowworm j's previous position, s is a step size constant, and $Distance_{jz}$ is the Euclidean distance between glowworms j and z.

B. Glowworm Swarm Optimization Clustering Algorithm - CGSO

In [10], the authors proposed a partitioning-based clustering algorithm (CGSO) by formulating the clustering problem as a multimodal optimization problem.

Given dataset D, a clustering algorithm extracts a set of clusters $C = \{C_1, C_2, ..., C_k\}$, each is represented with a point called centroid, such as $c = \{c_1, c_2, ..., c_k\}$, where k is the number of centroids in the c centroid set. Furthermore, the clustering algorithm tries to maximize the similarity of the instances in the same cluster, and to minimize the similarity of instances from different clusters.

In CGSO, the GSO objective is adjusted to locate multiple optimal centroids such that each centroid represents a sub-solution and the combination of these sub-solutions formulate the global solution for the clustering problem. The proposed CGSO consists of four main phases: initialization phase, luciferin level update, glowworm movement, and candidate centroids set construction.

In the initialization phase, first an initial glowworm swarm S of size m is created. For each glowworm g_j, a random position vector (p_i) is generated using uniform randomization within the given search space within the minimum and the maximum values that are calculated from the data set D. Then, the luciferin level (L_j) is initialized using the initial luciferin level L_0. The fitness function value F_j is initialized to zero. The local range r_s is set to an initial constant range r_0. Secondly, after initializing the swarm, the set of data instances cr_j which are covered by g_j, is extracted from data set D, and the $intraD_j$ is calculated using the following equation:

$$intraD_j = \sum_{i=1}^{|cr_j|} Distance(cr_{ji}, g_j) \tag{4}$$

where cr_{ji} is the data instance i which is covered by g_j; $|cr_j|$ is the number of data instances which is covered by g_j. Then, in the last step of the initialization phase, the Sum Squared Errors (SSE) and Inter Distance ($InterDist$) are calculated using Equation 5 and Equation 6, respectively.

$$SSE = \sum_{j=1}^{k} \sum_{i=1}^{|C_j|} (Distance(x_i, c_j))^2 \tag{5}$$

$$InterDist = \sum_{i=1}^{k} \sum_{j=i}^{k} (Distance(c_i, c_j))^2 \tag{6}$$

where the $Distance$ is calculated using the Euclidean distance.

After the initialization phase, an iterative process is performed to find optimal glowworms that represent the clustering problem centroids. The result of each iteration is an updated swarm with an updated candidate centroids set. In the luciferin level update phase, the fitness function F is evaluated using Equation 7 to assign new F_j values to each glowworm.

$$F(g_j) = \frac{\frac{1}{n}|cr_j|}{SSE \times \frac{intraD_j}{\max\limits_{j}\,(intraD_j)}} \tag{7}$$

After the fitness function evaluation for glowworm g_j, the luciferin level L_j is updated using Equation 1. Then, each glowworm g_j locates the neighborhood group to find the best neighbor. Then, the glowworm is moved towards the best neighbor by updating its p_j vector using the best neighbor position. After that, $|cr_j|$, and $intraD_j$ are updated based on the new glowworm g_j positions. After that, the candidate centroid set c is reconstructed based on the highest fitness values (F_j). The iterative process is continued until the size of the candidate centroid set c becomes less than a specific threshold (minimum number of centroids is given), or the maximum number of iterations is achieved. The candidate centroid set c decreases throughout the iterative process, and after the clustering process is completed, the candidate centroid set is used to evaluate the clustering results.

C. MapReduce Methodology

MapReduce is a highly scalable model and can be used across many computer nodes, and is mostly applicable for data intensive applications when there are limitations on multiprocessing and large shared-memory machines. MapReduce makes use of two main operations: Map and Reduce. Both Map and Reduce operations take inputs and produce outputs in the form of <key, value>. The Map operation goes over a large number of records and extracts interesting information from each record, and then all values with the same key are sent to the same Reduce operation. However, the Reduce operation aggregates intermediate results, generated from the Map function that has the same key, then generates the final results.

A well-known and commonly used implementation of MapReduce is Apache Hadoop [20]. It is an open source software framework that supports data-intensive distributed applications licensed under Apache. It enables applications to work with petabytes of data using thousands of independent processors. One of the main components of Hadoop is the storage component, Hadoop Distributed File System (HDFS). HDFS provides high-throughput access to the data and maintains fault tolerance by creating multiple replicas of the target data blocks. HDFS and MapReduce work together to support the ability of moving computation to the data, and not vice versa.

D. Proposed MRCGSO Approach

CGSO solves the clustering problem as an optimization task to obtain the best solution based on the coverage of centroids taking into consideration the intra-distance of each cluster. In CGSO, each glowworm competes to be a centroid and tries to cover the largest amount of data records, which means to have the highest coverage with the minimum intra-distance. Therefore, when the clustering is applied to big data, the calculation of the coverage and the intra-distance is very computationally expensive. MRCGSO solves this issue by parallelizing CGSO using the MapReduce methodology formulating CGSO with Map and Reduce functions.

In MRCGSO, each glowworm g_j in the swarm has the following information:

- Luciferin level (L_j)
- Fitness function value (F_j)
- M-dimensional position vector (p_j)
- Coverage set size (cr_j), which is the number of the data instances that are covered by g_j
- Intra-distance ($intraD_j$), which is the distance between the cr_j set members and the g_j position
- Local decision range (r_d), which is the range of the glowworm to find the covered data records within this range, and to find the glowworm's neighbors.

This information is updated in each iteration based on the previous swarm state. MRCGSO consists of three main steps, which are: initialization, coverage and distance computations, fitness evaluation and swarm movement. The initialization step includes a random initialization of the swarm members, and ensures that each glowworm has a position between the minimum and maximum values of the data set features. Moreover, each glowworm ensures to cover at least one data record, thus guaranteeing that the generated position of the glowworm is not an outlier, as well as ensuring that there is no empty candidate cluster.

In the coverage and distance step, the map and reduce functions are implemented based on the data records. After initialization, the swarm is written to the distributed cache. The mapper accesses the HDFS to read the data set, which is partitioned based on the number of mappers used. Each mapper retrieves the swarm from the distributed cache and then calculates the sub-covered records and at the same time the sub-intra-distances for each glowworm of the swarm based on the data chunk it holds. After that, the mapper will emit the result of each glowworm in the following <key, value> format, where the value is delimited by semicolons: <glowworm-id, sub-coverage;sub-intra-distance> where the sub-coverage is the number of data records covered by the glowworm, and the sub-intra-distance is the summation of the distance between the glowworm and the covered data records.

The reducer receives the intermediate output from the mappers and calculates the sum of sub-coverages and sub-intra-distances from all mappers in order to find the glowworm's cumulative coverage and cumulative intra-distance for the whole data set. Then, the reducer emits the results in the following <key, value> format: <glowworm-id, coverage;intra-distance> where the coverage is the number of data instances covered by the glowworm based on the whole data set, and the intra-distance is the sum of distances between the glowworm and the covered data instances from the whole data set. The reducers output will be saved on the HDFS. The number of mapper and reducers are multiple and are based on the data set size and the available hardware resources. The fitness evaluation and swarm movement step, reads the reducers' output and calculates the fitness of each glowworm, then continues the same CGSO process of updating the glowworm position based on its glowworm neighbors.

IV. Experiments and Results

In this section, we describe the experiments done to evaluate our proposed algorithm MRCGSO. First, the execution environment, and the information of the datasets used are given. Then, the experimental results are provided and discussed.

A. Environment and Datasets

We ran the experiments on one of the common Hadoop cluster that is used by researchers, the Longhorn Hadoop cluster hosted by the Texas Advanced Computing Center (TACC)[1]. The Longhorn Hadoop cluster has 48 nodes containing 48GB of RAM, 8 Intel Nehalem cores (2.5GHz each), which gives a total of 384 compute cores and 2.304 TB of aggregated memory. Experiments were performed using Hadoop version 0.20 (new API) for the MapReduce framework, and Java runtime 1.6 to implement the MRCGSO algorithm.

The MRCGSO settings were as recommended in [21]:

- $\rho = 0.4$
- $\gamma = 0.6$
- the initial luciferin level $L_0 = 5.0$
- the step size $s = 0.03$
- swarm size = 1000 glowworms
- maximum number of iterations = 100
- radial sensor range (local range) r_s is varied for different dataset as it depends on the data set, preliminary experiments were conducted to choose the best r_s value for each individual data set.

The real datasets that are used are the following:

- MAGIC: represents the results of registration simulation of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique. It was obtained from UCI machine learning repository[2].
- Electricity: contains electricity prices from the Australian New South Wales Electricity Market. The clustering process identifies two states (UP or DOWN) according to the change of the price relative to a moving average of the last 24 hours. Obtained from MOA[3].
- Poker Hand: is an examples of a hand consisting of five playing cards drawn from a standard deck of 52 cards. Each card is described using 10 attributes and the dataset describes 10 poker hand situations (clusters). It was obtained from UCI[2].
- Cover Type: represents cover type for 30 x 30 meter cells from US Forest. The real data set is obtained from the UCI[2]. It has 7 clusters that represent the type of trees.
- Synthetic: four series of datasets were generated using the data generator developed in [22]. The datasets range from 2 million to 16 million data records. In order to simplify the names of the synthetic datasets, we used names with specific pattern based on the data records number, the number of dimensions, and the number of the clusters. For example: the F2m2d5c dataset consists of 2 million $2m$ data records, each record is in 2 dimensions $2d$ and the dataset is distributed into 5 clusters $5c$.

[1]https://portal.longhorn.tacc.utexas.edu/
[2]http://archive.ics.uci.edu/ml/index.html
[3]http://moa.cs.waikato.ac.nz/datasets/

TABLE I
Summary of the datasets

Dataset	#Records	#Dim	Size (MB)	Type	#Clusters
MAGIC	$19,020$	10	3.0	*Real*	2
Electricity	$45,312$	8	6.0	*Real*	2
Poker	$1,025,010$	10	49.0	*Real*	10
CoverType	$581,012$	54	199.2	*Real*	7
F2m2d5c	$2,000,000$	2	83.01	*Synth*	5
F4m2d5c	$4,000,000$	2	165.0	*Synth*	5
F8m2d5c	$8,000,000$	2	330.3	*Synth*	5
F16m2d5c	$16,000,000$	2	660.4	*Synth*	5

B. Evaluation Measures

In our experiments, we used the parallel Speedup [23] measure to evaluate the performance of our MRCGSO algorithm, which is calculated using the following equation:

$$Speedup = \frac{T_2}{T_n} \tag{8}$$

where T_2 is the running time using 2 nodes, and T_n is the running time using n nodes, where n is a multiple of 2.

The speedup is obtained by fixing the swarm size while increasing the number of cluster nodes to evaluate the algorithm's ability to scale with increasing numbers of the cluster nodes. However, Scaleup is a measure of speedup that increases with increasing dataset sizes to evaluate the ability of the parallel algorithm utilizing the cluster nodes effectively, which is calculated using Equation 9.

$$Scaleup = \frac{T_{SN}}{T_{2SN}} \tag{9}$$

where T_{SN} is the running time for the dataset with size S using N nodes, and T_{2SN} is the running time using 2-fold of S and 2-folds of N nodes.

Moreover, we use the purity measure for the evaluation of the cluster quality [24], which is the standard measure of clustering quality and is calculated as:

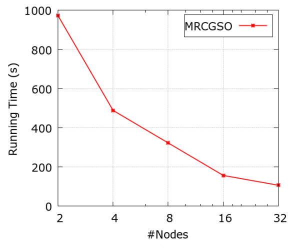
$$Purity = \frac{1}{n} \sum_{j=1}^{k} \max_i(\mid L_i \cap C_j \mid) \tag{10}$$

where L_i denotes the true assignments of the data instances in cluster i; q is the number of actual clusters in the data set.

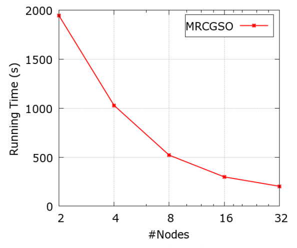
A clustering algorithm with large purity values indicates better clustering solutions. The clustering quality is perfect if the clusters only contain data instances from one true cluster; in that case the purity is equal to 1.

C. Results

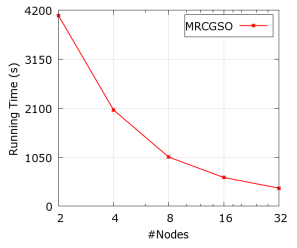
The running times and speedup measures for MRCGSO are shown in Figures 1 and 2, respectively. As can be noted from Figure 1, the improvement factor of MRCGSO's running times for the F2m2d5c, F4m2d5c, F8m2d5c, F16m2d5c data sets using 32 nodes are 9.09, 9.66, 10.54, 11.35, respectively,



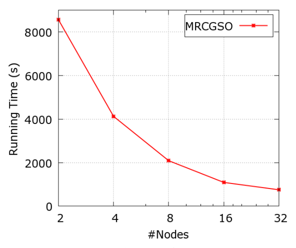
(a) F2m2d5c Time



(b) F4m2d5c Time

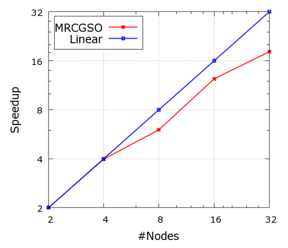


(c) F8m2d5c Time

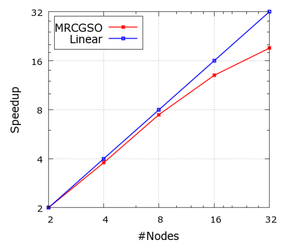


(d) F16m2d5c Time

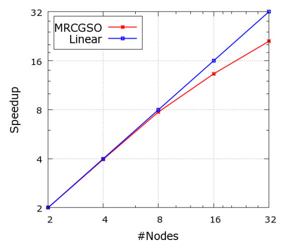
Fig. 1. Time Results



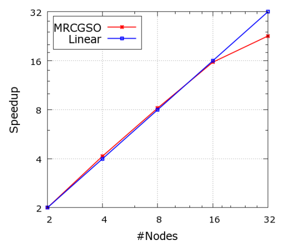
(a) F2m2d5c Speedup



(b) F4m2d5c Speedup



(c) F8m2d5c Speedup



(d) F16m2d5c Speedup

Fig. 2. Time and Speedup Results

compared to the running time with 2 nodes. The MRCGSO algorithm shows a significant improvement in running time. Furthermore, the running time of MRCGSO decreases almost linearly with increasing numbers of nodes of the Hadoop cluster. MRCGSO shows the same trend for all dataset sizes ranging from 2 million to 16 million.

In addition, the MRCGSO speedup in the Figure 2 scales close to linear (optimal speedup) for most data sets. It can be derived that for smaller datasets such as F2m2d4c, the speedup is good for 2 to 4 nodes then it drifts away from the linear. Whereas, for F4m2d5c, it is linear for 2, 4, and 8 nodes, and very close to 16 nodes and then drifts away. F8m2d5c shows the same results but has a closer speedup to linear for 16 nodes, while F16m2d5c has a linear speedup starting from 2 nodes to 16 nodes with a close value for 32 nodes. Therefore, we can conclude that the larger the dataset, the better the speedup. MRCGSO for F16m2d5c achieves a significant speedup that is very close to the linear speedup.

The results of the scaleup measure are shown in Figure 3, which are the MRCGSO results for increasing double folds of data set sizes (starting from 2, 4, 8 to 16 million data records) with the same double folds of nodes (2, 4, 8 to 16 nodes). The scaleup has almost a constant ratio and ranges between 1 and 1.12. The speedup for F2m2d5c is 1 while for F16m2d5c it is 1.12, which is a very small difference.

Experimental tests are implemented to test the purity of MRCGSO applied on four real data sets that vary in sizes (ranging between 19,020 and 2,025,010 records). The purity results are shown in Table II for the Magic, Electricity, Poker and Cover data sets; 0.66, 0.58, 0.53, and 0.55, respectively. These results outperform the purity of the known clustering algorithm K-Means, which achieves 0.60, 0.51, 0.11 and 0.32 respectively.

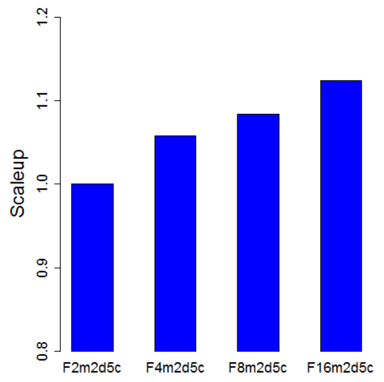


Fig. 3. Scaleup Measure

TABLE II
PURITY RESULTS

Dataset	**K-Means**	**MRCPSO**	**MRCGSO**
MAGIC	0.60	0.65	0.66
Electricity	0.51	0.58	0.58
Poker	0.11	0.51	0.53
CoverType	0.32	0.53	0.55

In addition, MRCGSO is compared with a MapReduce based clustering algorithm using PSO (MRCPSO) which was proposed in [25]. The purity results of MRCPSO are 0.65, 0.58, 0.51, and 0.53 for the same data sets, respectively. Comparing the purity results of MRCGSO and MRCPSO, we can see that MRCGSO has, as expected, comparable results for the four real data sets.

V. Conclusion

In this paper, we proposed a scalable design and implementation of a glowworm swarm optimization clustering (MR-CGSO) algorithm using MapReduce. The CGSO clustering algorithm is an effective method for data clustering, however, it needs a long time to process large data sets. Therefore, MR-CGSO was proposed to overcome the inefficiency of CGSO for big data sets. MRCGSO shows that CGSO can efficiently be parallelized with MapReduce to process very large data sets. In MRCGSO, the clustering task is formulated as a multimodal optimization to find the best centroids representing the clustering segments.

Experiments were conducted with millions of records showing good accuracy and good scalability. In addition, MRCGSO scales very well with increasing data set sizes and scales very close to the optimal speedup. Our future work aims to find a mechanism to select the optimal local decision range for the glowworms, and investigate if there is any relation between the range and the number of features of the data set and the data set size. Furthermore, we plan to apply MRCGSO in real application domains such as the community detection in the social networks.

Acknowledgment

The authors acknowledge the support of the ND EPSCoR IIP-DDA funding grant FAR0021976. In addition, this work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.

References

[1] P. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, 1st ed. Addison Wesley, May 2005.

[2] A. Engelbrecht, *Computational Intelligence, An Introduction*, second edition ed. Wiley, 2007.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE ICNN'95*. Australia, pp. 1942–1948, 1995.

[4] K. Krishnanand and D. Ghose, "Detection of multiple source locations using a glowworm metaphor with applications to collective robotics," in *IEEE Swarm Intelligence Symposium*, Pasadena, CA, USA, pp. 84 – 91, June 2005.

[5] C. Blum, "Ant colony optimization: Introduction and recent trends," *Physics of Life Reviews*, vol. 2, pp. 353–373, Dec. 2005.

[6] J. Handl, J. Knowles, and M. Dorigo, "Strategies for the increased robustness of ant-based clustering," in *Engineering Self-Organising Systems*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, vol. 2977, pp. 90–104, 2004.

[7] M. Omran, A. P. Engelbrecht, A. Salman, "Particle swarm optimization method for image clustering," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 3, pp. 297–322, 2009.

[8] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*. MIT Press Cambridge, MA, USA, 1995.

[9] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," in *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pp. 10–10, 2004.

[10] I. Aljarah and S. A. Ludwig, "A new clustering approach based on glowworm swarm optimization." in *IEEE Congress on Evolutionary Computation*. Mexico, Cancun: IEEE, pp. 2642–2649, 2013.

[11] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan, "Mr-dbscan: a scalable mapreduce-based dbscan algorithm for heavily skewed data," *Frontiers of Computer Science*, vol. 8, no. 1, pp. 83–99, 2014.

[12] Z. Weizhong, M. Huifang, and H. Qing, "Parallel k-means clustering based on mapreduce," in *Proceedings of the CloudCom'09*. Berlin, Heidelberg: Springer-Verlag, pp. 674–679, 2009.

[13] P. Zhau, J. Lei and W. Ye, "Large-scale data sets clustering based on mapreduce and hadoop," *Computational Information Systems*, vol. 7, no. 16, pp. 5956–5963, 2011.

[14] L. Guang, W. Gong-Qing, H. Xue-Gang, Z. Jing, L. Lian, and W. Xindong, "K-means clustering with bagging and mapreduce," in *Proceedings of the 2011 44th Hawaii International Conference on System Sciences*. Washington, DC, USA: IEEE Computer Society, pp. 1–8, 2011.

[15] S. Nair and J. Mehta, "Clustering with apache hadoop," in *Proceedings of the International Conference, Workshop on Emerging Trends in Technology*, ICWET'11. New York, NY, USA: ACM, pp. 505–509, 2011.

[16] S. Papadimitriou and J. Sun, "Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining," in *Proc. of the IEEE ICDM'08*, Washington, DC, USA, pp. 512–521, 2008.

[17] E. Alina, I. Sungjin, and M. Benjamin, "Fast clustering using mapreduce," in *Proceedings of KDD'11*. NY, USA: ACM, pp. 681–689, 2011.

[18] J. Yang and X. Li, "Mapreduce based method for big data semantic clustering," in *Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics*, SMC'13. Washington, DC, USA: IEEE Computer Society, pp. 2814–2819, 2013.

[19] F. Cordeiro, "Clustering very large multi-dimensional datasets with mapreduce," in *Proceedings of KDD'11*. NY, USA: ACM, pp. 690–698, 2011.

[20] (2012) Apache Software Foundation, Hadoop MapReduce [Online]. Available: http://hadoop.apache.org/mapreduce. [Online]. Available: http://hadoop.apache.org/mapreduce

[21] K. Krishnanand and D. Ghose, "Glowworm swarm optimisation: a new method for optimising multi-modal functions," *International Journal of Computational Intelligence Studies*, vol. 1, pp. 93–119, 2009.

[22] R. Orlandic, Y. Lai, and W. Yee, "Clustering high-dimensional data using an efficient and effective data space

reduction," in *Proc. ACM 14th Conf. on Information and Knowledge Management*, Bremen, Germany, pp. 201–208, 2005.

[23] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*. Addison-Wesley, USA, 2003.

[24] Y. Zhao and G. Karypis, "Evaluation of hierarchical clustering algorithms for document datasets," in *Proceedings of the eleventh CIKM'02*. NY, USA: ACM, pp. 515–524, 2002.

[25] I. Aljarah and S. A. Ludwig, "Parallel particle swarm optimization clustering algorithm based on mapreduce methodology," in *NaBIC*, Mexico, Mexico City, pp. 104–111, 2012.