# Parallel Particle Swarm Optimization Clustering Algorithm based on MapReduce Methodology

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

*Abstract*—**Large scale data sets are difficult to manage. Difficulties include capture, storage, search, analysis, and visualization of large data. In particular, clustering of large scale data has received considerable attention in the last few years and many application areas such as bioinformatics and social networking are in urgent need of scalable approaches. The new techniques need to make use of parallel computing concepts in order to be able to scale with increasing data set sizes. In this paper, we propose a parallel particle swarm optimization clustering (MR-CPSO) algorithm that is based on MapReduce. The experimental results reveal that MR-CPSO scales very well with increasing data set sizes and achieves a very close to the linear speedup while maintaining the clustering quality. The results also demonstrate that the proposed MR-CPSO algorithm can efficiently process large data sets on commodity hardware.**

***Keywords*—*Data Clustering, Parallel Processing, MapReduce, Hadoop***

## I. Introduction

Managing scientific data has been identified as one of the most important emerging needs of the scientific community in recent years. This is because of the sheer volume and increasing complexity of data being created or collected. In particular, in the growing field of computational science where increases in computer performance allow ever more realistic simulations and the potential to automatically explore large parameter spaces. As noted by Bell et al. [1]: "As simulations and experiments yield ever more data, a fourth paradigm is emerging, consisting of the techniques and technologies needed to perform data intensive science".

The question to address is how to effectively generate, manage and analyze the data and the resulting information. The solution requires a comprehensive, end-to-end approach that encompasses all the stages from the initial data acquisition to its final analysis.

Clustering [2] is a data mining technique used when analyzing data. The main goal of clustering algorithms is to divide a set of unlabeled data objects into different groups called clusters; each group has common specifications between the group members. The cluster membership measure is based on a similarity measure. To obtain high quality clusters, 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 very large data sets that contain large numbers of records with high dimensions is considered a very important issue nowadays. Examples are the clustering of profile pages in social networks, bioinformatics applications, and article clustering of big libraries. Most sequential clustering algorithms suffer from the problem that they do not scale with larger sizes of data sets, and most of them are computationally expensive in memory space and time complexities. For these reasons, the parallelization of the data clustering algorithms is paramount in order to deal with large scale data. To develop a good parallel clustering algorithm that takes big data into consideration, the algorithm should be efficient, scalable and obtain high quality clusters.

The MapReduce programming model [3], introduced by Google, has become very popular as an alternative model for data parallel programming over the past few years compared to the message passing methodology [4]. Large data clustering with MapReduce is an attractive solution to formulate the clustering algorithm that achieves high quality results within an acceptable amount of time. Furthermore, the parallelization with MapReduce is remarkable because it presents a programming model that parallelizes the tasks automatically, and provides fault-tolerance and load balancing.

Apart from Google's implementation of MapReduce, there are several popular open source implementations available such as Apache Hadoop MapReduce [5], and Disco [6]. MapReduce is a highly scalable model and can be used across many computer nodes. In addition, MapReduce is applicable when the target problem is considered data intensive and the computing environment has limitations on multiprocessing and large shared-memory machines. MapReduce moves the processing to the data and processes data sequentially to avoid random access that requires expensive seeks and disk throughput. MapReduce technologies have also been adopted by a growing number of groups in industry (e.g., Facebook [7], and Yahoo [8]. In academia, researchers are exploring the use of these paradigms for scientific computing, such as in the areas of Bioinformatics [9] and Geosciences [10] where codes are written using open source MapReduce tools.

The basic idea behind the MapReduce methodology is that the problem is formulated as a functional abstraction using two main operations: the *Map* operation and *Reduce* operation.

The *Map* operation iterates over a large number of records and extracts interesting information from each record, and all values with the same key are sent to the same *Reduce* operation. Furthermore, the *Reduce* operation aggregates intermediate results with the same key that is generated from the *Map* operation and then generates the final results. Figure 1 shows the MapReduce's main operations.

**Map Operation:**
*Map* (k, v) $\rightarrow$ [(k', v')]

**Reduce Operation:**
*Reduce* (k', [v']) $\rightarrow$ [(k', v')]

Fig. 1. The *Map* and *Reduce* Operations

Apache Hadoop [5] is the commonly used MapReduce implementation, and it is an open source software framework that supports data-intensive distributed applications licensed under Apache. It enables applications to work with thousands of computational independent computers and petabytes of data. Hadoop Distributed File System (HDFS - storage component) and MapReduce (processing component) are the main core components of Apache Hadoop. HDFS gives a high-throughput access to the data while maintaining fault tolerance by creating multiple replicas of the target data blocks. MapReduce is designed to work together with HDFS to provide the ability to move computation to the data not vice versa. Figure 2 shows the Hadoop architecture diagram with operation cycle. In this paper, we use the Hadoop framework for the MR-CPSO algorithm implementation.

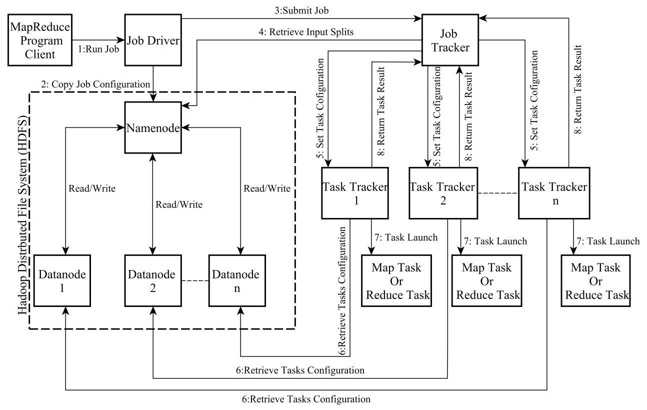


Fig. 2. Hadoop Architecture Diagram

This paper presents a parallel particle swarm optimization clustering (MR-CPSO) algorithm based on the MapReduce framework. In this work, we have made the following key contributions:

1) The proposed algorithm (MR-CPSO) makes use of the MapReduce framework that has been proven successful as a parallelization methodology for data-intensive applications [3, 5].
2) The proposed algorithm has been tested on large scale synthetic datasets with different sizes to show its speedup and scalability.
3) The proposed algorithm has been tested on real datasets with different settings to demonstrate its effectiveness and quality.

The rest of this paper is organized as follows: Section 2 presents the related work in the area of data clustering. In Section 3, the particle swarm optimization approach is introduced as well as our proposed MR-CPSO algorithm. Section 4 presents the experimental evaluation, and Section 5 presents our conclusions.

## II. Related Work

MapReduce has recently received a significant amount of attention in many computing fields but especially in the data mining area. Clustering has numerous applications and is becoming more challenging as the amount of data rises. Due to space constraints, we focus only on closely related work of parallel data clustering algorithms that employ the MapReduce methodology.

Zaho et al. in [11] proposed a parallel algorithm for k-means clustering based on MapReduce. Their algorithm randomly selects initial k objects as centroids. Then, centroids are calculated by the weighted average of the points within a cluster through the *Map* function; afterwards the *Reduce* function updates the centroids based on the distances between the data points and the previous centroids in order to obtain new centroids. Then, an iterative refinement technique is applied by MapReduce job iterations.

Li et al. in [12] proposed another MapReduce K-means clustering algorithm that uses the ensemble learning method bagging to solve the outlier problem. Their algorithm shows that the algorithm is efficient on large data sets with outliers.

Surl et al. [13] applied the MapReduce framework on co-clustering problems introducing a practical approach that scales well and achieves efficient performance with large datasets. The authors suggested that applying MapReduce on co-clustering mining tasks is very important, and discussed the advantages in many application areas such as collaborative filtering, text mining, etc. Their experiments were done using 3 real datasets and the results showed that the co-clustering with MapReduce can scale well with large data sets.

A fast clustering algorithm with constant factor approximation guarantee was proposed in [14], where they use sampling to decrease the data size and run a time consuming clustering algorithm such as local search on the resulting data set. A comparison of this algorithm with several sequential and parallel algorithms for the k-median problem was done using randomly generated data sets and a single machine where each machine used by the algorithms was simulated. The results showed that the proposed algorithm obtains better or similar solutions compared to the other algorithms. Moreover, the algorithm is faster than other parallel algorithms on very large data sets. However, for the k-median problem they have a small loss in performance.

In [15], the authors explored how to minimize the I/O cost for clustering with the MapReduce model and tried to minimize the network cost among the processing nodes. The proposed technique BOW (Best Of both Worlds), is a subspace clustering method to handle very large datasets in efficient time and derived its cost functions that allow the automatic, dynamic differentiation between disk delay and network delay. Experiments on real and synthetic data of millions of points with good speedup results were reported.

In this paper, the clustering task is expressed as an optimization problem to obtain the best solution based on the minimum distances between the data points and the cluster centroids. For this task, we used Particle Swarm Optimization (PSO) [16] as it performs a globalized search to find the best solution for the clustering task problem (this solves the K-means [17, 2] sensitivity of the selection of the initial cluster centroids and avoids the local optima convergence problem). PSO is one of the common optimization techniques that iteratively proceeds to find the best solution based on a specific measure.

PSO has been used to solve a clustering task in [18], where the problem discussed was document clustering. The authors compared their results with K-Means, whereby the PSO algorithm proved to generate more compact clustering results. However, in this paper we are validating this approach with more generalized and much larger datasets. In addition, the MapReduce framework has been chosen as the parallelization technique in order to tackle the computational and space complexities that large datasets incur causing an efficiency degradation of the clustering.

To the best of our knowledge, this is the first work that implements PSO clustering with MapReduce. Our goal is to show that PSO clustering benefits from the MapReduce framework and works on large datasets achieving high clustering quality, scalability, and a very good speedup.

## III. Proposed Approach

### A. Introduction to Particle Swarm Optimization

Particle swarm optimization (PSO) is a swarm intelligence method first introduced by Kennedy and Eberhart in 1995 [16]. The behavior of particle swarm optimization is inspired by bird flocks searching for optimal food sources, where the direction in which a bird moves is influenced by its current movement, the best food source it ever experienced, and the best food source any bird in the flock ever experienced. In PSO, the problem solutions, called particles, move through the search space by following the best particles. The movement of a particle is affected by its inertia, its personal best position, and the global best position. A swarm consists of multiple particles, each particle has a fitness value which is assigned by the objective function to be optimized based on its position. Furthermore, a particle contains other information besides the fitness value and position such as the velocity which direct the moving of the particle. In addition, PSO maintains the best personal position with the best fitness value the particle has ever seen. Also, PSO holds the best global position with the best fitness value any particle has ever experienced. Many variants of PSO were introduced in literature. In our work, the Global Best Particle Swarm Optimization [16, 19] variant is used.

The following equations are used to move the particles inside the problem search space:

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{1}$$

where $X_i$ is the position of particle $i$, $t$ is the iteration number and $V_i$ is the velocity of particle $i$.

PSO uses the following equation to update the particle velocities, also used in our proposed algorithm:

$$\begin{aligned} V_i(t+1) = W \cdot V_i(t) + (r_1 \cdot cons_1) \cdot [XP_i - X_i(t)] \\ +(r_2 \cdot cons_2) \cdot [XG - X_i(t)] \end{aligned} \tag{2}$$

where $W$ is inertia weight, $r_1$ and $r_2$ are randomly generated numbers, $cons_1$, $cons_2$ are constant coefficients, $XP_i$ is the current best position of particle $i$ and $XG$ is the current best global position for the whole swarm.

### B. Proposed MapReduce PSO Clustering Algorithm (MR-CPSO)

In the MR-CPSO algorithm, we formulated the clustering task as an optimization problem to obtain the best solution based on the minimum distances between the data points and the cluster centroids. The MR-CPSO is a partitioning clustering algorithm similar to the k-means clustering approach, in which a cluster is represented by its centroid. In k-means clustering, the centroid is calculated by the weighted average of the points within a cluster. In MR-CPSO, the centroid for each cluster is updated based on the swarm particles' velocities.

In MR-CPSO, each particle $P_i$ contains information which is used in the clustering process such as:

- Centroids Vector ($CV$): Current cluster centroids vector.
- Velocities Vector ($VV$): Current velocities vector.
- Fitness Value ($FV$): Current fitness value for the particle at iteration $t$.
- Best Personal Centroids ($BPC$): Best personal centroids seen so far for $P_i$.
- Best Personal Fitness Value ($BPC_{FV}$): Best personal fitness value seen so far for $P_i$.
- Best Global Centroids ($BGC$): Best global centroids seen so far for whole swarm.
- Best Global Fitness Value ($BGC_{FV}$): Best global fitness value seen so far for whole swarm.

This information is updated in each iteration based on the previous swarm state.

In MR-CPSO, two main operations need to be adapted and implemented to apply the clustering task on large scale data: the fitness evaluation, and particle centroids updating.

Particle centroids updating is based on PSO movement Equations 1 and 2 that calculate the the new centroids in each iteration for the individual particles. The particle centroids update takes a long time, especially when the particle swarm size is large.

Besides the update of the particle centroids, the fitness evaluations are based on a fitness function that measures the distance between all data points and particle centroids by taking the average distance between the particle centroids. The fitness evaluation is based on the following equation:

$$Fitness = \frac{\sum_{j=1}^{k} \frac{\sum_{i=1}^{n_j} Distance(R_i, C_j)}{n_j}}{k} \tag{3}$$

where $n_j$ denotes the number of records that belong to cluster $j$; $R_i$ is the $i^{th}$ record; $k$ is the number of available clusters; $Distance(R_i, C_j)$ is the distance between record $R_i$ and the cluster centroid $C_j$. In this paper, we use the Manhattan distance applying the following equation:

$$Distance(R_i, C_j) = \sum_{v=1}^{D} |R_{iv} - C_{jv}| \tag{4}$$

where $D$ is the dimension of record $R_i$; $R_{iv}$ is the value of dimension $v$ in record $R_i$; $C_{jv}$ is the value of dimension $v$ in centroid $C_j$.

The fitness evaluation takes a long time to execute when working with large data sets. For example, if the data set contains 5 million data points with 10 dimensions, and the number of clusters is 5, the swarm size is 30, then the algorithm needs to calculate $5 \times 10^6 \times 5 \times 10 \times 30 = 75 \times 10^8$ distance values for one iteration. This takes 400 minutes running on a 3.2 GHz processor.

Figure 3 shows the MR-CPSO architecture diagram.

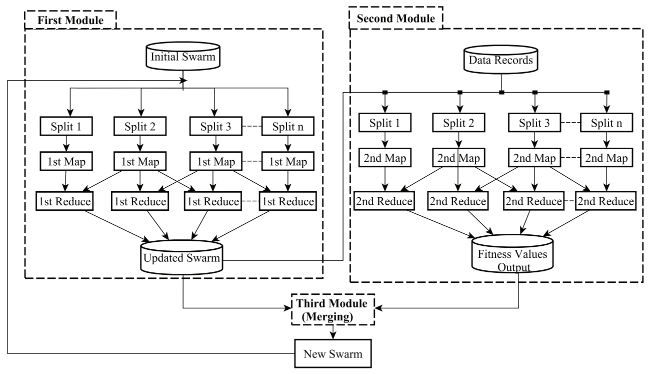


Fig. 3. MR-CPSO Algorithm Architecture Diagram

The MR-CPSO algorithm consists of three main sub-modules:

- The first module is a MapReduce job to update the particle swarm centroids.
- The second module is a MapReduce job for the fitness evaluation of the swarm with new particle centroids that are generated in the first module.
- The third (merging) module is used to merge the fitness values calculated from the second module with the updated swarm which is generated in the first module. Also, in this module, the best personal centroids and best global centroids are updated. Afterwards, the new particles are ready for the next iteration.

*1) First Module:* In the first module, the MapReduce job is launched for updating the particle centroids. The *Map* function receives the particles with identification numbers. However, the particle ID represents the *Map* key and the particle itself represents the value. The *Map* value contains all information about the particle such as $CV$, $VV$, $FV$, $BPC$ and $BGC$, which are used inside the *Map* function. In the *Map* function, the centroids are updated based on the PSO equations 1 and 2. The other information such as PSO coefficients ($cons_1$,$cons_2$), inertia weight ($W$), which are used in the PSO equations, are retrieved from the job configuration file. After that, the *Map* function emits the particle with updated centroids to the *Reduce* function. To benefit from the MapReduce framework, we use the number of *Maps* relative to the number of cluster nodes and swarm size. The *Reduce* function in the first module is only an identity reduce function that is used to sort the *Map* results and combine all of them into one output file. Furthermore, the particle swarm is saved in the distributed file system to be used by two other modules. The pseudo-code of the *Map* function and *Reduce* function is shown in Figure 4.

```
function Map (Key: particleID, Value: Particle)
 Initialization:
 $particleID$=$Key$
 $particle$=$Value$
 //Extract the information from the particle
 extractInfo($CV$,$VV$,$BPC$,$BGC$)
 Generate random numbers $r1$ and $r2$
 for each $c_i$ in $CV$ do
   //update particle velocity
   for each $j$ in $Dimension$ do
     $newVV_{ij}$= $w$ * $VV_{ij}$
                 +($r1$ * $cons_1$)*($BPC$ - $c_{ij}$)
                 +($r2$ * $cons_2$)*($BGC$ - $c_{ij}$)
     $newc_{ij}$=$c_{ij}$ + $newVV_{ij}$
   end for
   update($particle$,$newVV_i$, $newc_i$)
end for
Emit($particleID$, $particle$)
 end function

function Reduce (Key: ParticleID, ValList: Particle)
 for each $Value$ in ValList do
   Emit($Key$, $Value$)
end for
end function
```

Fig. 4. Algorithm - First Module

*2) Second Module:* In the second module, the MapReduce job is launched to calculate the new fitness values for the updated swarm. The *Map* function receives the data records with recordID numbers. The recordID represents the *Map* key and the data record itself represents the value. The *Map* and *Reduce* functions work as shown in the Figure 5 outlining the pseudo code of the second module algorithm. The *Map* function process starts with retrieving the particle swarm from the distributed cache, which is a feature provided by the MapReduce framework for caching files. Then, for each particle, the *Map* function extracts the centroids vector and calculates the distance value between the record and the centroids vector returning the minimum distance with its centroidID. The *Map* function uses the ParticleID with its centroidID that has the minimum distance to formulate a new composite key. Also, a new value is formulated from the minimum distance. After that, the *Map* function emits the new key and new value to the *Reduce* function.

The *Reduce* function aggregates the values with the same key to calculate the average distances and assigns it as a fitness value for each centroid in each particle. Then, the *Reduce* function emits the key with average distance to formulate the new fitness values. Then, the new fitness values are stored in the distributed file system.

```
function Map (key: RecordID, Value: Record)
 Initialization:
 $RID$=$key$
 $record$=$value$
 //Read the particles swarm from the Distributed Cache
 read($Swarm$)

 for each $particle$ in $Swarm$
   $CV$=extractCentroids(particle)
   $PID$=extractPID(particle)
   $minDist$=returnMinDistance($record$,$CV$)
   $centroidID$=$i$
//$i^{th}$ centroid contains minDist
   $newKey$=($PID$,$centroidID$)
   $newValue$=($minDist$)
   Emit($newKey$, $newValue$)
 end for
end function

function Reduce (Key:(PID,centerId),ValList:(minDist,1))
 Initialization:
 $count$=0, $sumDist$=0, $avgDist$=0
 for each $Value$ in ValList
   $minDist$=extractminDist( $Value$ )
   $count$=$count + 1$
   $sumDist$=$sumDist$ + $minDist$
 end for
 $avgDist$=$sumDist$ / $count$
 Emit($Key$, $avgDist$)
end function
```

Fig. 5. Algorithm - Second Module

*3) Third Module (Merging):* In the third module of the MR-CPSO algorithm, the main goal is to merge the outputs of the first and second modules in order to have a single new swarm. The new fitness value (FV) is calculated on the particle level by a summation over all centroids' fitness values generated by the second module. After that, the swarm is updated with the new fitness values. Then, $BPC_{FV}$ for each particle is compared with the new particle fitness value. If the new particle fitness value is less than the current $BPC_{FV}$, $BPC_{FV}$ and its centroids are updated. Also, the $BGC_{FV}$ with centroids is updated if there is any particle's fitness value smaller than the current $BGC_{FV}$. Then, the new swarm with new information is saved in the distributed file system to be used as input for the next iteration.

## IV. Experiments and Results

In this section, we describe the clustering quality and discuss the running time of the measurements for our proposed algorithm. We focus on scalability as well as the speedup and the clustering quality.

### A. Environment

We ran the MR-CPSO experiments on the Longhorn Hadoop cluster hosted by the Texas Advanced Computing Center (TACC)[1] and on our NDSU[2] Hadoop cluster. The Longhorn Hadoop cluster is one of the common Hadoop cluster that is used by researchers. The Longhorn Hadoop cluster contains 384 compute cores and 2.304 TB of aggregated memory. The Longhorn Hadoop cluster has 48 nodes containing 48GB of RAM, 8 Intel Nehalem cores (2.5GHz each), whereas our NDSU Hadoop cluster consists of only 18 nodes containing 6GB of RAM, 4 Intel cores (2.67GHz each) with HDFS 2.86 TB aggregated capacity. For our experiments, we used Hadoop version 0.20 (new API) for the MapReduce framework, and Java runtime 1.6 to implement the MR-CPSO algorithm.

TABLE I
Summary of the datasets

| Dataset | #Records | #Dim | Size (MB) | Type | #Clusters |
|---|---|---|---|---|---|
| TwoEllipses | $2,000$ | 2 | 0.14 | *Real* | 2 |
| FourCircles | $2,000$ | 2 | 0.14 | *Real* | 4 |
| 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 | *Synthetic* | 5 |
| F4m2d5c | $4,000,000$ | 2 | 165.0 | *Synthetic* | 5 |
| F6m2d5c | $6,000,000$ | 2 | 247.6 | *Synthetic* | 5 |
| F8m2d5c | $8,000,000$ | 2 | 330.3 | *Synthetic* | 5 |
| F10m2d5c | $10,000,000$ | 2 | 412.6 | *Synthetic* | 5 |
| F12m2d5c | $12,000,000$ | 2 | 495.0 | *Synthetic* | 5 |
| F14m2d5c | $14,000,000$ | 2 | 577.9 | *Synthetic* | 5 |
| F16m2d5c | $16,000,000$ | 2 | 660.4 | *Synthetic* | 5 |
| F18m2d5c | $18,000,000$ | 2 | 743.6 | *Synthetic* | 5 |
| F30m2d5c | $30,000,000$ | 2 | 1238.3 | *Synthetic* | 5 |
| F32m2d5c | $32,000,000$ | 2 | 1320.8 | *Synthetic* | 5 |

### B. Datasets

To evaluate our MR-CPSO algorithm, we used both real and synthetic datasets as described in Table I. The real datasets that are used are the following:

[1]https://portal.longhorn.tacc.utexas.edu/
[2]http://www.ndsu.edu

- Two Ellipses: contains point coordinates in 2 dimensions. The data set contains 2 balanced clusters, where each cluster formulates an ellipse.
- Four Circles: contains point coordinates in 2 dimensions. The data set contains 4 balanced clusters, where each cluster formulates a circle.
- 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[4].
- 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[5].
- 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[4].
- Cover Type: represents cover type for 30 x 30 meter cells from US Forest. The real data set is obtained from the UCI[4]. It has 7 clusters that represent the type of trees.
- Synthetic: Two series of datasets with different sizes of records were generated using the data generator developed in [20]. The first series are 9 datasets ranging from 2 million to 18 million data records. The second series are 2 datasets with 30 million and 32 million data records. In order to simplify the names of the synthetic datasets, the datasets' names consist of the 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 data records, each record is in 2 dimensions, and the dataset is distributed into 5 clusters.

## C. Evaluation Measures

In our experiments, we used the parallel Scaleup [21] and Speedup [21] measures calculated using Equations 5 and 6, respectively. These measures are used to evaluate the performance of our MR-CPSO algorithm. 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.

$$Scaleup = \frac{T_{SN}}{T_{2SN}} \tag{5}$$

where the $T_{SN}$ 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.

For the Speedup measurement, the dataset is fixed and the number of cluster nodes is increased by a certain ratio.

[4]http://archive.ics.uci.edu/ml/index.html

[5]http://moa.cs.waikato.ac.nz/datasets/

$$Speedup = \frac{T_N}{T_{2N}} \tag{6}$$

where the $T_N$ is the running time using $N$ nodes and $T_{2N}$ is the running time using 2-fold of $N$ nodes.

We evaluate the Scaleup by increasing the dataset sizes and number of cluster nodes with the same ratio.

For the clustering quality, we used the purity measure [2] in Equation 7 to evaluate the MR-CPSO clustering correctness.

$$Purity = \frac{1}{N} \times \sum_{i=1}^{k} max_j(\mid C_i \cap L_j \mid) \tag{7}$$

where $C_i$ contains all the points assigned to cluster $i$ by MR-CPSO, and $L_j$ denotes the true assignments of the points in cluster $j$; $N$ is the number of records in the dataset.

We used the PSO settings that are recommended by [22, 23]. We used a swarm size of 100 particles and inertia weight $W$ of 0.72. Also, we set the acceleration coefficient constants $cons_1$ and $cons_2$ to 1.7.

Figure 6 shows the MR-CPSO clustering quality results visualized for the TwoEllipses and FourCircles datasets. The results show that the MR-CPSO algorithm is able to assign the data records to the correct cluster, with purity 1.0 for the TwoEllipses and purity 0.997 for the FourCircles dataset.

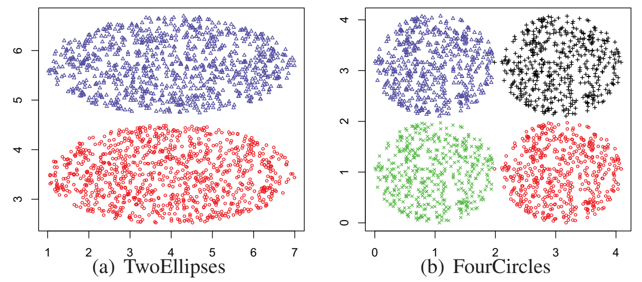


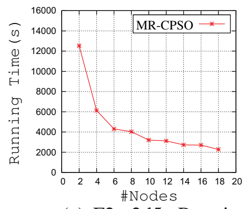
Fig. 6. MR-CPSO Clustering Results. 6(a) clustering results for the TwoEllipses dataset with Purity=1.0. 6(b) clustering results for the FourCircles dataset with Purity=0.997.

## D. Results

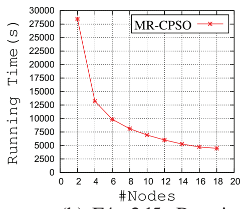
We used the real datasets to evaluate the correctness of the MR-CPSO algorithm. We compared the purity results of MR-CPSO with those of the standard K-means algorithm, which is implemented in the Weka data mining software [24], in order to perform a fair comparison of the purity values. The maximum iterations used for K-means and MR-CPSO is 25.

The purity results of MR-CPSO for the TwoEllipses, FourCircles, MAGIC, Electricity, Poker, Cover Type datasets are *1.0, 0.997, 0.65, 0.58, 0.51, and 0.53*, respectively. On the other hand, K-means results are *1.0, 1.0, 0.60, 0.51, 0.11, and 0.32*, respectively. We can observe that the purity of the MR-CPSO clustering results after 25 iterations are better than the K-means clustering results.

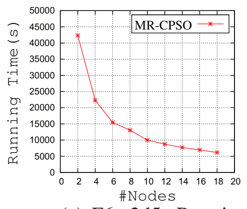
We used MR-CPSO for clustering different sizes of synthetic datasets. We ran MR-CPSO with 18 NDSU cluster nodes by increasing the number of nodes in each run by multiples of 2. In each run, we report the running time and speedup



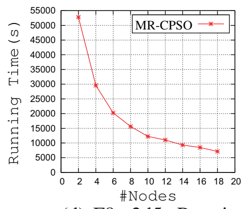
(a) F2m2d5c Running Time



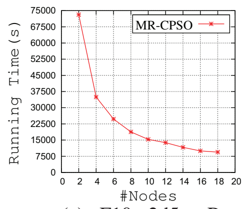
(b) F4m2d5c Running Time



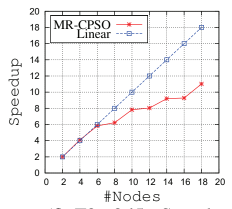
(c) F6m2d5c Running Time



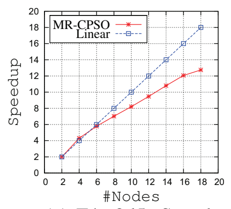
(d) F8m2d5c Running Time



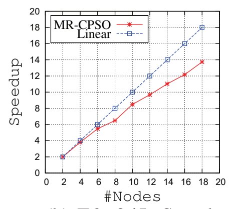
(e) F10m2d5c Running Time



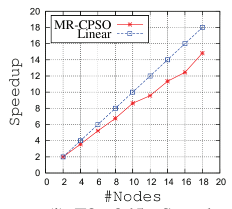
(f) F2m2d5c Speedup



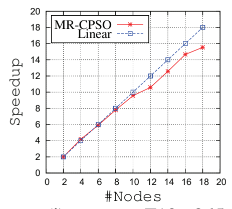
(g) F4m2d5c Speedup



(h) F6m2d5c Speedup

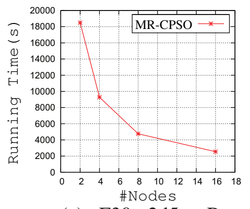


(i) F8m2d5c Speedup

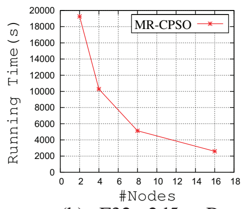


(j) F10m2d5c Speedup

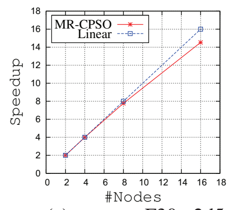
Fig. 7. Running time and speedup results on the synthetic datasets with 18 NDSU Hadoop cluster nodes and 25 iterations of MR-PSO. 7(a) - 7(e) Running time for synthetic datasets from 2 million to 10 million data records. 7(f) - 7(j) Speedup measure for synthetic datasets from 2 million to 10 million data records.



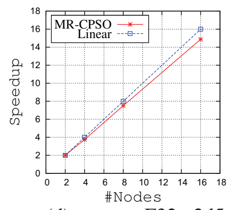
(a) F30m2d5c Running Time



(b) F32m2d5c Runnimg Time



(c) F30m2d5c Speedup



(d) F32m2d5c Speedup

Fig. 8. Running time and Speedup results on the synthetic datasets with Longhorn Hadoop cluster and 10 iterations of MR-PSO. 8(a), 8(b) Running times for synthetic datasets for 30 million and 32 million data records. 8(c), 8(d) Speedup measure for synthetic datasets for 30 million and 32 million data records.

of 25 iterations of MR-PSO. The running times and speedup measures are shown in Figure 7. As can be noted from the figure, the improvement factor of MR-CPSO's running times for the F2m2d5c, F4m2d5c, F6m2d5c, F8m2d5c, F10m2d5c datasets using 18 nodes are 5.5, 6.4, 6.9, 7.4, 7.8, respectively, compared to the running time with 2 nodes. The MR-CPSO algorithm demonstrates a significant improvement in running time. Furthermore, the running time of MR-CPSO decreases almost linearly with increasing number of nodes of the Hadoop cluster. In addition, the MR-CPSO speedup scales close to linear for most datasets. MR-CPSO algorithm with F10m2d5c achieves a significant speedup obtaining very close to the linear speedup.

Figure 8 shows the running time and speedup of MR-CPSO with larger datasets using the Longhorn Hadoop cluster. We used 16 nodes as the maximum number of nodes to evaluate the MR-CPSO performance since we have limited resources on the Longhorn cluster. The running time results for the two data sets decreases when the number of nodes of the Hadoop cluster increases. The improvement factor of MR-CPSO running times for the F30m2d5c and F32m2d5c datasets with 18 nodes are 7.27, 7.43 compared to the running time with 2 nodes. The MR-CPSO algorithm shows a significant improvement in running time. The MR-CPSO algorithm with F30m2d5c and F32m2d5c achieve a significant speedup which is almost identical to the linear speedup. Thus, if we want to cluster even larger datasets with the MR-CPSO algorithm, we can accomplish that with a good performance by adding nodes to the Hadoop cluster.

Figure 9 shows the Scaleup measure of MR-CPSO for increasing double folds of data set sizes (starting from 2, 4, 6, 8 to 18 million data records) with the same double folds of nodes (2, 4, 6, 8 to 18 nodes), implemented on the NDSU Hadoop cluster. Scaleup for F4m2d5c was 0.85, and it captures almost a constant ratio between 0.8 and 0.78 when we increase the number of available nodes and dataset sizes with same ratio.

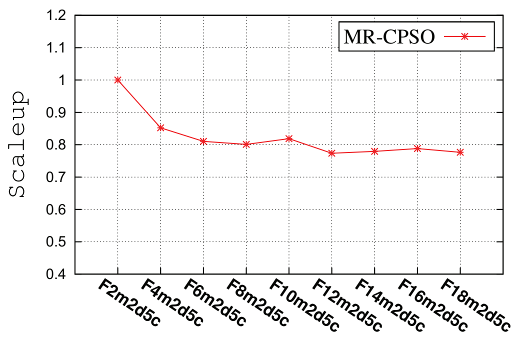


Fig. 9. MR-CPSO Scale-up

## V. Conclusion

In this paper, we proposed a scalable MR-CPSO algorithm using the MapReduce parallel methodology to overcome the inefficiency of PSO clustering for large data sets. We have shown that the MR-CPSO algorithm can be successfully parallelized with the MapReduce methodology running on commodity hardware. The clustering task in MR-CPSO is formulated as an optimization problem to obtain the best solution based on the minimum distances between the data points and the cluster centroids. The MR-CPSO is a partitioning clustering algorithm similar to the k-means clustering approach, in which a cluster is represented by its centroid. The centroid for each cluster is updated based on the particles' velocities.

Experiments were conducted with both real-world and synthetic data sets in order to measure the scaleup and speedup of our algorithm. The results reveal that MR-CPSO scales very well with increasing data set sizes, and scales very close to the linear speedup while maintaining good clustering quality. The results also show that the clustering using MapReduce is better than the K-means sequential algorithm in terms of clustering quality.

Our future plan is to include measurements with different dimensions as well as even larger data sets. Also, we will investigate the impact of the PSO settings on the cluster quality. Furthermore, we plan to apply the MR-CPSO algorithm on some massive application such as intrusion detection.

## Acknowledgment

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.

## References

[1] G. Bell, A. Hey, and A. Szalay, "Beyond the data deluge," *Science 323 AAAS*, vol. 39, 2006.

[2] J. Han, *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco, CA, USA, 2005.

[3] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," 2004, pp. 137–150. [Online]. Available: http://www.usenix.org/events/osdi04/tech/dean.html

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

[5] (2011) Apache software foundation, hadoop mapreduce. [Online]. Available: http://hadoop.apache.org/mapreduce

[6] (2011) Disco mapreduce framework. [Online]. Available: http://discoproject.org

[7] (2011) Hadoop - facebook engg, note. [Online]. Available: http://www.facebook.com/note.php?noteid=16121578919

[8] (2011) Yahoo inc. hadoop at yahoo! [Online]. Available: http://developer.yahoo.com/hadoop

[9] T. Gunarathne, T. Wu, J. Qiu, and G. Fox, "Cloud computing paradigms for pleasingly parallel biomedical applications," in *Proceedings of 19th ACM International Symposium on High Performance Distributed Computing*. ACM, January 2010, pp. 460–469.

[10] S. Krishnan, C. Baru, and C. Crosby, "Evaluation of mapreduce for gridding lidar data," in *Proceedings of the CLOUDCOM '10*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 33–40.

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

[12] 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, 2011, pp. 1–8.

[13] 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 '8*, Washington, DC, USA, 2008, pp. 512–521.

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

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

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

[17] A. Jain, M. Murty, and P. Flynn, "Data clustering: A review." NY, USA: ACM, 1999.

[18] X. Cui, T. Potok, and P. Palathingal, "Document clustering using particle swarm optimization," in *IEEE Swarm Intelligence Symposium*. Pasadena, California, USA: ACM, 2005, pp. 185–191.

[19] T. Schoene, S. A. Ludwig, and R. Spiteri, "Step-optimized particle swarm optimization," in *Proceedings of the 2012 IEEE CEC*. Brisbane, Australia, June 2012.

[20] 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, 2005, pp. 201–208.

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

[22] H. Shi and R. Eberhart, "Parameter selection in particle swarm optimization," in *Proc. 7th Annual Conference on Evolutionary Programming*, San Diego, CA, 1998, pp. 201–208.

[23] I. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," in *Information Processing Letters*, 2003.

[24] I. Witten, E. Frank, and M. Hall, *Data Mining: Practical Machine Learning Tools And Techniques, 3rd Edition*. Morgan Kaufmann, 2011.