Chapter 5
Spark-Based Design of Clustering Using Particle Swarm Optimization

Mariem Moslah, Mohamed Aymen Ben HajKacem, and Nadia Essoussi

5.1 Introduction

Large volume of data are being collected from different sources and there is a high demand for methods that can efficiently analyze such data referred to as Big data analysis. Big data usually refers to three main dimensions, also called the three Vs [11], which are, respectively, *Volume*, *Variety*, and *Velocity*. Volume refers to the large amount of data, Variety refers to the number of types of data, and Velocity refers to the speed of data processing. Hence, exploring and organizing large-scale data using machine learning techniques becomes an important challenge in Big data analysis.

Clustering is an important technique in machine learning which has been used to look for hidden models, relations, or to summarize data. Technically, clustering aims to organize data into a predetermined number of groups where objects within the same group share some common characteristics. Examples of clustering methods categories are hierarchical methods, density-based methods, grid-based methods, model-based methods, and partitional methods [18]. K-means [14] as one of the partitional clustering methods, it remains the most efficient because of its simplicity and its low computational complexity. However, it is sensitive to the selection of initial cluster centers, as it may converge to suboptimal solutions if the initial cluster centers are not properly chosen [5].

To overcome this weakness, several optimization algorithms were introduced to perform data clustering. Genetic algorithm (GA) which is based on a mutation operator for clustering analysis was proposed [12]. Another approach based on simulated annealing (SA) for data clustering was proposed [3] and more recently

M. Moslah · M. A. B. HajKacem (✉) · N. Essoussi
LARODEC, Institut Supérieur de Gestion de Tunis, Université de Tunis, Le Bardo, Tunisia
e-mail: nadia.essoussi@isg.rnu.tn

O. Nasraoui, C.-E. Ben N'Cir (eds.), *Clustering Methods for Big Data Analytics*, Unsupervised and Semi-Supervised Learning,
https://doi.org/10.1007/978-3-319-97864-2_5

the particle swarm optimization (PSO) was proposed for data clustering [17]. Among these algorithms, particle swarm optimization (PSO), as one of the swarm intelligence algorithms, has gain a great popularity in the last two decades and seemed to be potentially full and fertile research area [15]. In addition, PSO algorithm does not require high computational capacities or a lot of parameters to adjust, compared to genetic algorithms [8]. Although the efficiency of PSO algorithm for data clustering, it does not scale with the increasing volume of data. This is explained by the high computation time to build grouping from large amount of data.

To deal with large-scale data, Aljarah and Ludwig [2] proposed fitting PSO clustering into MapReduce model. However, this method has some considerable shortcomings. The first shortcoming is the result of inherent conflict between MapReduce and PSO clustering. PSO is an iterative algorithm and it requires to perform many iterations for producing optimal results. In contrast, MapReduce has a significant problem with iterative algorithms. At each iteration, the whole data set must be read and written to disks and this results a high input/output (I/O) operations. This significantly degrades the performance of MapReduce-based method. The second shortcoming is inherited from the PSO clustering algorithm. PSO suffers from a low convergence speed when it approaches the global optimum region. According to [1], particles tend to move slowly when they reach the convergence region.

To deal with these issues, we propose in this chapter a new **S**park-based **PSO** clustering method, referred to as S-PSO. First, we propose to exploit the benefits provided by Spark, by using in-memory operations to reduce the efficiency of existing MapReduce solutions. Second, we propose a modified version of PSO which executes k-means algorithm when it approaches the global optimum region to accelerate the convergence. The rest of this chapter is organized as follows: Sect. 5.2 presents a background about the basic concepts related to the particle swarm optimization algorithm, MapReduce model, and Spark framework. In Sect. 5.3, existing works related to data clustering using PSO and Big data clustering methods are presented. Section 5.4 presents our proposed method S-PSO. Section 5.5 offers a theoretical analysis of the proposed method. Section 5.6 presents experiments that we have performed to evaluate the efficiency of the proposed method. Section 5.7 presents conclusion and future works.

5.2 Background

This section first presents the particle swarm optimization, followed by the Big data technologies used in this work.

5.2.1 Particle Swarm Optimization

Particle swarm optimization (PSO) was first introduced in 1995 by Kennedy the social psychologist and Eberhart the electrical engineer. At first the algorithm was intended to simulate the social behavior of birds when searching for food. When a bird recognizes a food area, it broadcasts the information to all the swarm. Hence, all the birds follow him and this way they raise the probability of finding the food since it is a collaborative work. Then the behavior of birds within swarms was turned into an intelligent algorithm capable of solving several optimization problems [15].

PSO is a population-based optimization algorithm. It is composed of a swarm of particles where each particle is considered as a potential solution to the optimization problem. Each particle i is characterized at the time t, by the current position $x_i(t)$ in the search space, the velocity $v_i(t)$, the personal best position $pbestP_i(t)$, and fitness value $pbestF_i(t)$. The personal best position represents the best position that the particle has ever achieved throughout its movement, which is defined as follows:

$$pbestP_i(t+1) = \begin{cases} pbestP_i(t)\ if\ \ f(pbestP_i(t)) <= f(x_i(t+1)) \\ x_i(t+1)\ if\ \ f(pbestP_i(t)) > f(x_i(t+1)) \end{cases} \tag{5.1}$$

The swarm's best position $gbestP(t)$ corresponds to the best position that the whole swarm has ever seen, which is calculated as follows:

$$gbestP(t+1) = \min\ (f(y), f(gbestP(t))) \tag{5.2}$$

where $y \in \{pbestP_0(t), ..., pbestP_S(t)\}$. Particle's positions and velocities are calculated using the following equations:

$$x_i(t+1) \leftarrow x_i(t) + v_i(t) \tag{5.3}$$

$$v_i(t+1) \leftarrow wv_i(t) + c_1r_1(pbestP_i(t) - x_i(t)) + c_2r_2(gbestP(t) - x_i(t)) \tag{5.4}$$

where w is referred to the inertia weight, $x_i(t)$ is the position of the particle i at the time t, $v_i(t)$ is the velocity of the particle i at the time t, c_1 and c_2 are two acceleration coefficients, and r_1 and r_2 are two random values in the range [0,1]. The main algorithm of PSO is described in Algorithm 1 [10].

Algorithm 1 The main algorithm of PSO

```
1: Input: Z: input data set
2: Output: Particles information
3: Initialize the swarm of particles from Z.
4: while Convergences not reached do
5:     Compute the fitness of particles according to the fitness function to be optimized.
6:     Update each particle's personal best position and fitness value using Equation 5.1.
7:     Update the global best position using Equation 5.2.
8:     Update particle's positions and velocities using Equation 5.3 and 5.4 respectively.
9: end while
```

It is important to note that the convergence is reached when $gbest$ does not have significant changes anymore [15].

5.2.2 *MapReduce Model*

MapReduce is a very well-known programming framework built to ensure the parallel computation and processing of large volume of data. It adopts the method of divide and conquer in which a problem is divided into smaller and less complex sub-problems. Then simultaneously, all sub-problems are separately executed. Once finished, results are merged to provide a final solution to the very big and complex problem [6]. The principal components of the MapReduce model are the map and reduce functions. The map function takes as input key/value pairs (k, v), performs the assigned work, and generates intermediate key/value pairs $[(k', v')]$. An intermediate step known as shuffling step is required to organize each intermediate key with its corresponding values [7].

The reduce function aims to merge all the values corresponding to each intermediate key $(k',[v'])$ to form the final result and output final key/value pairs as $[(k' ,v'')]$[7].

Figure 5.1 outlines the flowchart of MapReduce paradigm. The enormous data set is divided into several chunks, small enough to be fitted into a single machine, each chunk is then assigned to a map function to be processed in parallel. Inputs and outputs are stored in the distributed file system and are accessible by each node in the cluster. Apache Hadoop is the most popular implementation of MapReduce for Big data processing and storage on commodity hardware. The use of this framework has

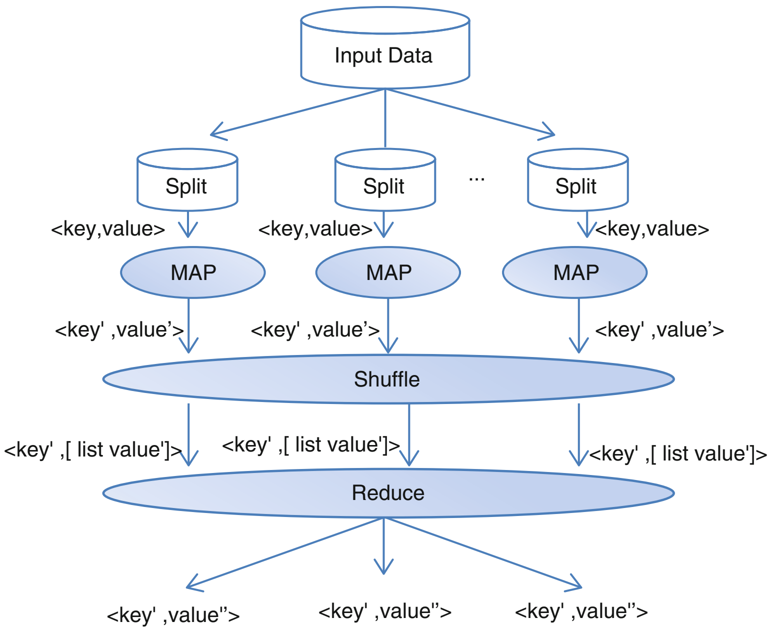

Fig. 5.1 MapReduce flowchart

become widespread in many fields because of its performance, open source nature, installation facilities, and its distributed file system named Hadoop distributed file system (HDFS). In spite of its great popularity, Hadoop MapReduce has significant problems with iterative algorithms.

5.2.3 Apache Spark

Apache Spark, a new framework for Big data processing, designed to be able to solve the Hadoop's shortcomings. It was developed in 2009 in AMPLab in UC Berkeley University, then it became open source in 2010. It is known for its scalability, flexibility, and rapidity [16]. Spark gains its prosperity from its capacity of performing in-memory computations which means that data does not have to be moving from and to the disk instead it is maintained in the memory. In fact, Spark loads the necessary data needed for a specific application, processes and keeps resulted data in memory for further iterations or processes. Therefore, data are read and written only once, rather than multiple times, when iterative processes are required. It also supports streaming processing and this is considered as a strength point regarding Hadoop framework. Spark is based on resilient distributed dataset (RDD), which is considered as a database table that is distributed among the different nodes of the cluster. The RDDs could be created by reading an external data source or by parallelizing a collection of data. Two major operations could be performed on RDDs, namely *transformations* and *actions*. Transformations apply changes on an RDD and generate a new one. Examples of transformations are Map, Sample, GroupbyKey, and ReducebyKey. Actions are operations that require the generation of an output. Examples of actions are Count, Take, and Foreach.

5.3 Related Works

PSO is widely used in cluster analysis since it uses a whole population of possible solutions that collaborate in the purpose of finding better quality clusters [1, 9, 17]. Merwe and Engelbrecht [17] are the first to propose clustering using particle swarm optimization algorithm. In fact, they proposed two methods for data clustering which are the PSO clustering algorithm and the hybrid method which combines both PSO and k-means and then they compared it with k-means algorithm. PSO clustering method is based on the particle swarm optimization algorithm which constitutes a population of particles considered as potential solutions to the clustering problem, these particles contain each k randomly selected initial centroids. This approach is different from that of k-means because it considers a whole swarm of solutions and not only one as in k-means. The particles move in the search space until a global optimal solution is reached. The hybrid method uses the result of k-means algorithm as one of the initial solutions, then randomly generates the rest of the swarm to finally execute the PSO clustering algorithm. The two approaches were compared with the stand-alone k-means algorithm and they showed significantly better performances in convergence and quantization error. Esmin et al. [9] proposed a clustering method based upon the method of Merwe and Engelbrecht [17] with modifications on the regular fitness function. Since the fitness computation doesn't take into account the number of data vectors in each cluster, result's quality was significantly influenced. The number of data objects within each cluster was introduced in the fitness computation which results in good improvements. Ahmadyfard and Modares [1] proposed a new hybrid PSO and K-means clustering algorithm. It starts by executing PSO and switches to k-means when the algorithm reaches the global optimum region. The switch is obtained when the fitness function remains significantly unchanged after several iterations. This combination takes advantage of the strength points of both k-means and PSO and in the same time overcomes their weaknesses. Since PSO shows poor convergence speed near optimum, it is then combined with k-means to speed up the convergence. This combination brings significant improvement compared with the stand-alone PSO and k-means algorithms. Despite the efficiency of the latter discussed methods to deal with the initialization problem using PSO, they are not able to scale with huge volume of data.

To deal with large-scale data, several methods which are based on parallel frameworks have been designed in the literature [2, 4, 13, 20]. Most of these methods use the MapReduce model for data processing. For instance, Zaho el al. [20] have implemented k-means method through MapReduce model. This method first assigns each data point to the nearest cluster prototypes in the map function. The reduce function then updates the new cluster prototypes. Then, this method iterates calling the two functions several times until convergence. Ben HajKacem et al. have proposed fitting k-prototypes using MapReduce model [4] in order to perform the clustering of mixed large-scale data. This method iterates two main steps until convergence: the step of assigning each data point to the nearest cluster center and the step of updating cluster centers. These two steps are implemented in map and reduce phase, respectively. Ludwing has proposed the parallelization of fuzzy c-means clustering using MapReduce model [13]. This method is based on two MapReduce jobs. The first MapReduce job calculates the cluster centers by scanning the input data set. The second MapReduce job also iterates over the data set and calculates the distances to be used to update the membership matrix as well as to calculate the fitness. Although the performance of the latter discussed methods to deal with large-scale data using MapReduce, they do not provide a solution regarding the sensitivity to the selection of the initial cluster centers.

Aljarah and Ludwig [2] proposed MR-CPSO which is, to the best of our knowledge, the only method that processes large-scale data clustering using MapReduce model as well as PSO to ensure better scalability and better clustering quality. In order to perform clustering using PSO, few steps have to be performed which start by the particles initialization, fitness update, centroids update, and finally personal best and global best update. The proposed implementation based on MapReduce suggests three major modules. The first module is a map reduce module responsible for centroids update. The map function receives the particles information as a key value pair where the key is the particle ID and the value represents all the information related to the particle. The map function extracts from each particle the necessary information that enables the update of the centroids and that is done using the position and velocity update formulas (1.3) and (1.4). Then the reduce function combines all updated information into one file and load it into the distributed file system. Once the centroids are updated, the algorithm launches the second module. The second module is a MapReduce module where the map function performs the data assignment step and the reduce function updates the fitness values for each particle. In fact, the map function retrieves the updated particles information and then receives the data chunk to use, then for each particle, data objects are assigned to the closest center. Once done, a key/value pair is generated where the key represents the particle ID and centroid ID where the data is assigned and the value represents the computed distance. Then, the reduce function uses the generated key/value pairs from the previous map function to compute the new fitness values that have to be stored in the distributed file system. The last module is a simple module that merges the outputs resulted from the different previous modules. In addition to that, personal best are updated for each particle as well as a global best. The updated particle's information are stored in the distributed file system to be used for next iterations. Figure 5.2 presents the modules of MR-CPSO.

Table 5.1 summarizes the existing methods.

Although the attested performance of MR-CPSO to perform large-scale data, it has some considerable shortcomings.

- PSO suffers from a low convergence speed close to the global optimum region. In fact, according to [1], particles tend to move slowly when they reach the convergence region. This could be explained by the fact that in PSO, particles tend to follow the global best one and when the algorithm is about to converge, all the particles are almost in the same convergence region. Therefore every

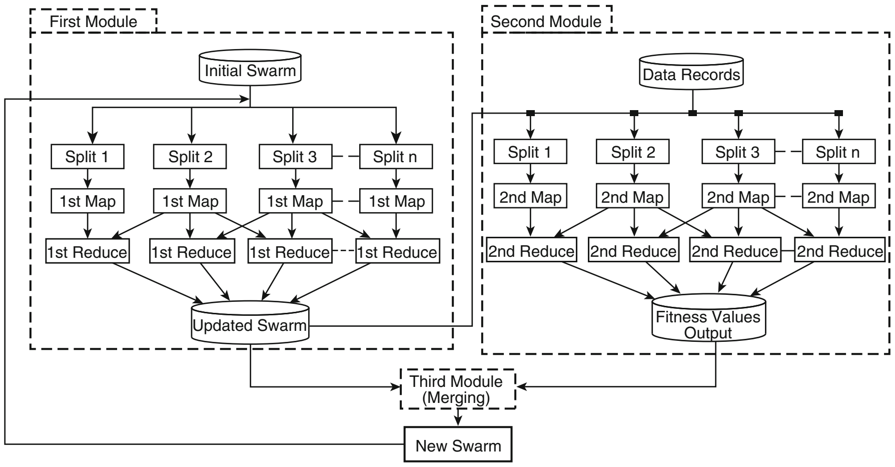

Fig. 5.2 MR-CPSO modules

Table 5.1 Summary of existing methods

	Initialization	Scalability
Merwe and Engelbrecht [17]	+	−
Esmin et al. [9]	+	−
Ahmadyfard and Modares [1]	+	−
Zhao et al. [20]	−	+
Ben HajKacem et al. [4]	−	+
Ludwing [4]	−	+
Aljarah and Ludwig [2]	+	+

particle's $pbest P_i$ and $x_i(t)$ are almost equal to the $gbest P$. Therefore, the particles velocities and positions are not exhibiting significant changes.

- PSO is an iterative algorithm and it requires to perform some iterations for producing optimal results. In contrast, MapReduce has a significant problem with iterative algorithms. As a consequence, the whole data set must be loaded from the file system into the main memory at each iteration. Then, after it is processed, the output must be written to the file system again. Therefore, many of I/O operations like I/O disk occur during each iteration and this decelerates the running time.

5.4 Proposed Approach: S-PSO for Clustering Large-Scale Data

We propose a new efficient PSO clustering method using Spark. The proposed method S-PSO is based on a new strategy that runs k-means algorithm in the latest stages. In fact, k-means is a very fast algorithm so it will accelerate the convergence and it will not affect the final result quality since PSO is very close to the convergence. Furthermore, the proposed method reads the data set only once in contrast to existing MapReduce implementation of PSO clustering. Hence, we aim to exploit in our implementation the flexibility provided by Spark framework, by using in-memory operations that alleviate the consumption time of existing MapReduce solution [2].

S-PSO method is composed of four major steps, namely *Data assignment and fitness computation step*, *Personal and global best update step*, *Position and velocity update step*, and finally *K-means iteration step*. The main process of the proposed method denoted by S-PSO is described in Fig. 5.3.

5.4.1 Data Assignment and Fitness Computation Step

S-PSO starts by setting an initial swarm where it initializes every particle's position, velocity, personal best position, and personal best fitness. The positions are the initial cluster's centroids and they are randomly retrieved from the data. Therefore, each particle once initialized represents a possible solution of the data clustering.

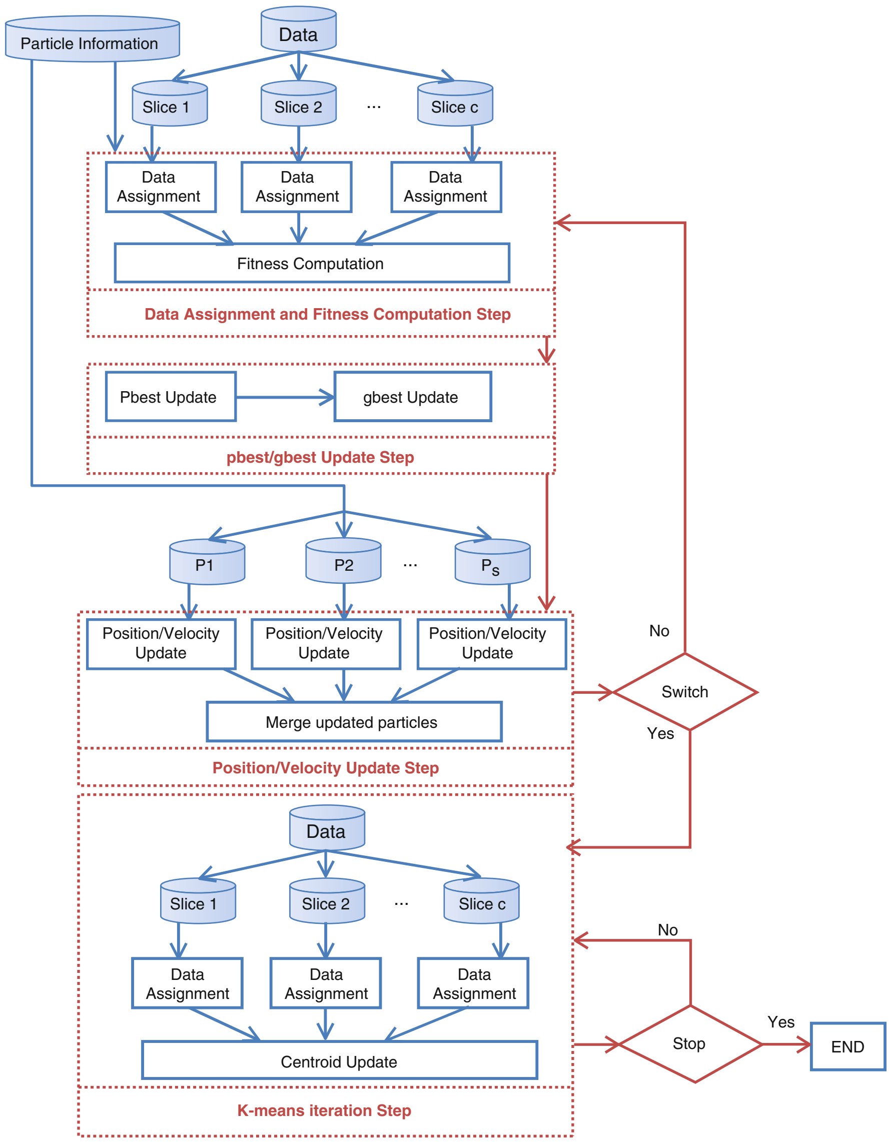

Fig. 5.3 Flowchart S-PSO

This initial swarm encompasses particle's information that will be used for the remaining steps. The data assignment step is a highly expensive operation because it requires to assign the huge amount of data to their closest clusters and this has to be done for every single particle. Since assigning an object is independent from the other objects therefore this step could be performed in parallel. First, the data set is divided into chunks and every chunk is assigned to a map function along with the particle's information. The map function, called *Data assignment*, assigns the data point from its corresponding chunk to the closest cluster in each particle. Then, the map function returns as output a key value pair where the key is composed of the couple particleID and centroidID and the value designates the minimum distance between a data object and the centroidID in a specific particleID.

Once all the data are already assigned to the closest cluster, a reduce function, called *fitness computation* step uses the *reduceByKey()* operation provided by Spark framework to combine the different outputs from the different map functions. The reduce function computes the new fitness value using for that the quantization error given by the following formula:

$$F_i = \frac{\sum_{j=1}^{k} [\sum_{\forall z_p \in C_{ij}} d(z_p, C_j)/|C_{ij}|]}{k} \tag{5.5}$$

where $d(z_p, C_j)$ represents the distance between the data object z_p and the cluster's centroid C_j, $|C_{ij}|$ represents the number of objects assigned to the centroid C_{ij} relative to the particle i, and finally k represents the number of clusters.

Then, the reduce function provides as output a key value pair composed of the particleID as a key and the new fitness value as the value. Let $Z = \{z_1 ... z_n\}$ the input data set. Let $P(t) = \{P_i(t)...P_S(t)\}$ the set of the particle's information where $P_i(t) = \{x_i(t), v_i(t), pbestP_i(t), pbestF_i(t)\}$ represents the information of particle i in the iteration t where $x_i(t)$ is the position, $v_i(t)$ is the velocity, $pbestP_i(t)$ is the best position, and $pbestF_i(t)$ is the best fitness.

Let $F = \{F_1...F_S\}$ the set of fitness values where F_i is the fitness value of the particle i.

Algorithm 2 outlines the data assignment and fitness computation step.

Algorithm 2 Data assignment and fitness computation step

```
1:  Input: Z: input data set, P(t): particle information
2:  Output: F: fitness values
3:  Split the data set Z into c chunks $Z = \{Z^1...Z^c\}$
4:  % Map Phase
    Let $Z^j$ be assigned to map j
5:  for each $z_p \in Z^j$ do
6:      for each $P_i(t) \in P(t)$ do
7:          $x_i(t) \leftarrow$ Extract position from $P_i(t)$
8:          Assign data objects to their closest centroid by computing the euclidean distance
9:          Let distance the minimum computed distance
10:         Let CentroidID the index of the centroid where the object $z_p$ is assigned
11:         Let ParticleID be the index of the particle i
12:     end for
13:     Emit (key: ParticleID, CentroidID /value: distance)
14: end for
15: % Reduce Phase
16: for each $P_i(t) \in P$ do
17:     Compute fitness value $F_i$ using Equation 5.5
18:     Emit (key: ParticleID /value: $F_i$)
19: end for
```

5.4.2 Pbest and Gbest Update Step

When the new particle's fitness is computed, it is automatically stored in an RDD distributed among the cluster's nodes. However, since Pbest and gbest update is not an expensive step and it does not require to be performed in parallel that's why the RDD containing the particle's fitness is collected which means it is returned to the driver program to be used in a serial way. Now, each particle updates its personal best position. For the gbest update, the particle having the best fitness value (the smallest quantization error) is identified as gbest particle.

Let $pbestF(t) = \{pbestF_1(t)...pbestF_S(t)\}$ the set of personal best fitness values where $pbestF_i(t)$ is the pbestF of the particle i at iteration t.
Let $pbestP(t) = \{pbestP_1(t)...pbestP_S(t)\}$ the set of personal best position where $pbestP_1(t)$ is the pbestP of the particle i at iteration t. Let $gbestP$ be the position of the best particle.
Algorithm 3 outlines the pbest and gbest update step.

Algorithm 3 Pbest and gbest update step

```
1: **Input:** F, $pbestF(t)$, $pbestP(t)$
2: **Output:** $pbestF(t+1)$, $pbestP(t+1)$, $gbestP$
3: $gbestP \leftarrow \emptyset$
4: **for** each particle $P_i(t) \in P(t)$ **do**
5:     $pbestF_i(t+1) \leftarrow \emptyset$
6:     $pbestP_i(t+1) \leftarrow \emptyset$
7:     **if** $(pbestF_i(t) \leq F_i)$ **then**
8:         $pbestF_i(t+1) \leftarrow pbestF_i(t)$
9:         $pbestP_i(t+1) \leftarrow pbestP_i(t)$
10:    **else**
11:        $pbestF_i(t+1) \leftarrow F_i$
12:        $pbestP_i(t+1) \leftarrow x_i(t+1)$
13:    **end if**
14: **end for**
15: Let $i^*$ is the index of particle having the best fitness value
16: $gbestP \leftarrow x_{i^*}(t)$
```

5.4.3 Position and Velocity Update Step

To take advantage of the parallel environment, S-PSO starts by splitting the particles information among different map functions to perform the velocity and position update using Eqs. (5.3) and (5.4).

Then, the reduce function merges the results provided from the different map functions into one single RDD using for that a *reduceByKey()* operation.

Once finished, the data set and the particle's information stored both in RDDs are persisted in memory for the next iteration and are not returned to the disk. The persistence is performed using the operation *cache()* or *persist()*.

Let $x(t) = \{x_1(t)...x_S(t)\}$ the set of position values where $x_i(t)$ is the position of the particle i at iteration t. Let $v(t) = \{v_1(t)...v_S(t)\}$ the set of velocity values where $v_i(t)$ is the velocity of the particle i at iteration t.

Algorithm 4 outlines the position and velocity update step.

Algorithm 4 Position and velocity update step

```
 1: Input: $gbestP$, $P(t)$
 2: Output: P(t+1)
 3: % Map Phase
    Let $P_i(t)$ be assigned to a map function i
 4: $x_i(t+1) \leftarrow \emptyset$
 5: $v_i(t+1) \leftarrow \emptyset$
 6: Compute the new position value $x_i(t+1)$ using 5.4
 7: Compute the new velocity value $v_i(t+1)$ using 5.3
 8: Emit( key: 1/ value: $P_i(t+1)$)
 9: % Reduce Phase
10: Merge outputs of the different map functions
11: Emit ($P(t+1)$)
```

S-PSO continues iterating until it almost reaches the global optimum region where it becomes very slow. In order to overcome this problem, when the S-PSO reaches the switch condition, it automatically switches to k-means algorithm to take advantage of its speed.

The switch is realized when the variable Time-To-Start is reached: it determines the iteration number where the switch has to occur.

5.4.4 K-Means Iteration Step

When the algorithm switches to k-means, it takes as input the final global best position retrieved from PSO to serve as an initial cluster centroid. K-means is composed of two major steps: data assignment and centroids update. The data assignment step is a map function that takes as input the data chunk and the initial clusters centroids, then it assigns the data objects to the closest cluster. For that, it generates as output a list of key/value pair where key represents the index of the cluster where the data object is assigned and the value represents the data vector.

The centroid update step is a reduce function responsible for merging the different outputs of the map function and for updating the clusters centroids using the mean operation in each cluster. K-means iteration step iterates until it reaches the maximal number of iterations.

Let $C(t) = \{c_1(t)...c_k(t)\}$ the set of cluster centroids at iteration t. Algorithm 5 outlines the k-means iteration step.

Algorithm 5 K-means iteration step

```
1:  Input: Z, $gbestP$
2:  Output: C(t+1)
3:  Split the data set Z into c chunks $Z = \{Z^1 ... Z^c\}$
4:  C(t) $\leftarrow$ $gbestP$
5:  % Map Phase
    Let $Z^j$ assigned to map j
6:  for each $z_p \in Z^j$ do
7:      for each $C_i(t) \in C(t)$ do
8:          Assign data objects to their closest centroid by computing the euclidean distance
9:          Let CentroidID the index of the centroid where the object $z_p$ is assigned
10:     end for
11:     Emit (key:CentroidID /value: $z_p$)
12: end for
13: % Reduce Phase
14: for each $C_i(t) \in C(t)$ do
15:     Update centroid $C_i(t+1)$
16:     Emit (key: CentroidID /value: $C_i(t+1)$)
17: end for
```

Algorithm 6 outlines the overall steps of S-PSO.

Algorithm 6 S-PSO algorithm

```
1:  Input: Z: input data set, Iter: maximal iteration number, S: swarm size, k: number of clusters,
    Time-To-Start: iteration number for switching to K-means, P(t): initial swarm's information
2:  Output: $C_f$: Final centroids
3:  i $\leftarrow$ 1
4:  switch $\leftarrow$ false
5:  while (( i< Iter) and (!switch)) do
6:      % Data assignment and fitness computation step
7:      % Pbest and gbest Update step
8:      % Position and velocity update step
9:      i++
10:     if ( i =Time-To-Start) then
11:         switch $\leftarrow$ true
12:         $C = gbestP$
13:     end if
14: end while
15: if ( switch =true) then
16:     j $\leftarrow$ 1
17:     repeat
18:         % K-means iteration step
19:         j $\leftarrow$ j + 1
20:     until $(j = Iter)$
21: end if
```

5.5 Theoretical Analysis

5.5.1 Complexity Analysis

Complexity analysis aims to provide the time, space, and I/O complexities of our proposed method S-PSO and compares it to MR-CPSO proposed by [2] since it is the only work dealing with Big data clustering using PSO.

The following notations are used: n the data set size, k the number of clusters, c the number of data chunks, I the number of iterations, s the swarm size, and P the size of the list containing the particle's information.

5.5.1.1 Time Complexity

The most expensive operation in PSO is the data assignment step where each data object has to compute its distance to all the clusters of each particle in the swarm, then this has to be repeated several times. Therefore, the time complexity of PSO could be estimated to $O(n.k.s.I)$.

The S-PSO splits the input data into several chunks that could be processed simultaneously. So instead of processing n data object in every iteration which is the case for PSO, S-PSO will use n/c data item per iteration. Hence, at each iteration S-PSO takes $O(n/c.k.s)$ time. Similar to S-PSO, MR-CPSO works on chunks, thus it takes $O(n/c.k.s)$ time for each iteration. However, regarding the number of iterations, MR-CPSO executes PSO for I times while for S-PSO the number of iteration is divided into I_1 for running PSO and I_2 for running k-means where I_1+I_2=I (I_1, I_2 $< I$). Since K-means is a very fast algorithm which means that executing it for I_2 times will definitely reduce the overall consumed time.

Therefore, the overall time complexity for S-PSO is estimated to $O(n/c.k.s.I_1 + n/c.k.I_2)$. While the overall time complexity for MR-CPSO is estimated to $O(n/c.k.s.I)$. Hence, S-PSO decreases the time complexity of MR-CPSO from $O(n/c.k.s.I)$ to $O(n/c.k.s.I_1 + n/c.k.I_2)$ where $I_1, I_2 < I$.

5.5.1.2 Space Complexity

The S-PSO and MR-CPSO store in the memory a data set of size n where it is used in the data assignment and fitness computation step. In addition to that they initialize and store a list containing the particles information. Therefore, the space complexity of S-PSO and MR-CPSO is estimated to $O(n + P)$.

5.5.1.3 Input/Output Complexity

S-PSO reads the data chunk from disk only once and persists it in the memory. Therefore, the I/O complexity of S-PSO is evaluated by $O(n/c)$. While the MR-

CPSO has to access the disk I times corresponding to the number of iterations. Hence, the I/O cost of MR-CPSO is evaluated by $O(n/c.I)$. As a result, S-PSO can reduce the I/O complexity of MR-CPSO from $O(n/c.I)$ to $O(n/c)$.

5.5.2 Time-To-Start Variable Analysis

In PSO, when particles approach the global optimum region, they tend to become very slow. In order to speed up the convergence, k-means known for its speed was integrated in the latest stages of PSO. This way, S-PSO starts by processing PSO first, then it switches to k-means once it approaches the convergence area. We aim by this combination to take advantage of PSO's capacity of finding good quality results on one hand and on the other hand take advantage of k-mean's speed. The switching between the two algorithms is conditioned by a variable that we introduce called Time-To-Start. Time-To-Start designates the iteration number where the switch has to occur. The choice of this variable can influence both time and quality. If this variable is picked up to be in the beginning of PSO, we obtain a low quality result but a very reduced execution time and the opposite is correct. Therefore, this variable has to be chosen in a way that ensures balance between quality and time.

5.6 Experiments and Results

5.6.1 Methodology

In order to evaluate the efficiency of S-PSO method, we performed experiments that aim to figure out three major points. (i) How efficient S-PSO method is when applied to large-scale data compared to existing methods? (ii) How the Time-To-Start variable can improve the performance of the proposed method? (iii) How the Spark framework can enhance the scalability of the proposed method when dealing with large-scale data?

5.6.2 Environment and Data Sets Description

Experiments were realized using a machine of 16 GB of RAM and $1T$ of disk having 8 cores, it uses Apache Spark version 2.1.1 and scala version 2.1.1 running on a Ubuntu version 16.04. We conducted the experiments on the following data sets:

- Simulated data set: four series of large-scale data sets are generated using the gaussian distribution where the mean is 350 and the sigma is 100. The data sets range from 1 million to 8 millions data points. Each data point is described using

10 attributes. The numeric values are generated with gaussian distribution. In order to simplify the names of the simulated data sets, we use the notations D1M, D2M, D4M, and D8M to denote a simulated data set containing 1, 2, 4, and 8 millions data points, respectively.

- Magic: is a real data set which represents the results of registration simulation of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique. The magic data set contains 19,020 instances having each 10 attributes. The clustering process for this data set identifies whether the energy registered is gamma or not. This data set was obtained from UCI machine learning repository.[1]
- KDD Cup data set (KDD): is a real data set which consists of normal and attack connections simulated in a military network environment. The KDD data set contains about 5 millions connections. Each connection is described using 33 attributes. The clustering process for this data set detects the type of the attacks among all connections. This data set was obtained from UCI machine learning repository. [2]
- Household data set (House) : is a real data set which represents the results of measurements of electric power consumption in household. The House data set contains 2,075,259 data points. Each data point is described using 10 attributes. The clustering process for this data set identifies the types of electric consumption in household. This data set was obtained from UCI machine learning repository.[3]
- CoverType: is a real data set that represents cover type for 30×30 meter cells from US Forest. CoverType data set contains 581,012 instances having each 54 attributes. This data set helps to predict the type of the tree from 7 different types. The real data set is obtained from the UCI[4]. Statistics of these data sets are summarized in Table 5.2.[4]

Table 5.2 Summary of the data sets

Data set	Number of data points	Number of attributes	Domain
D1M	1,000,000	10	Simulated
D2M	2,000,000	10	Simulated
D4M	4,000,000	10	Simulated
D8M	8,000,000	10	Simulated
Magic	19,020	10	Gamma particle's energy
KDD	4,898,431	33	Intrusion detection
House	2,075,259	10	Electricity
CoverType	581,012	54	Agriculture

[1]https://archive.ics.uci.edu/ml/machine-learning-databases/magic/.

[2]https://archive.ics.uci.edu/ml/machine-learning-databases/kddcup99-mld/.

[3]https://archive.ics.uci.edu/ml/machine-learning-databases/00235/.

[4]https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/.

5.6.3 Performance Measures

Evaluating the performance of our proposed method is an important step that takes into account evaluation measures in order to assess the algorithm's quality. The evaluation is addressed towards the approach's scalability and robustness in addition to the quality of clustering results.

- In order to evaluate the scalability, we use the speedup, scaleup, and sizeup measures, which are defined as follows.
  - The speedup measure consists of fixing the data set size and varying the number of computer nodes. Therefore, clustering is performed with one computer node then with m computer nodes, having respectively T_1 and T_m as running time [19]. The speed up is measured as follows:

$$\text{Speedup} = T_1/T_m \tag{5.6}$$

    The more the speedup is close to linear, the best the algorithm is.
  - While the speedup measure keeps the data set size constant and increases the computer nodes, the scaleup measure increases both the data set size and the number of computer nodes. Therefore, the clustering is performed with one computer node and s as data set size, then it is performed with m nodes with $m * s$ as a data set size [19]. The scaleup measure is given by the following formula:

$$\text{Scaleup} = T_{1s}/T_{ms} \tag{5.7}$$

    The algorithm is considered scalable when T_{1s} is almost equal to T_{m*s}.
  - The sizeup measure holds constant the number of computer nodes and increases m times the size of the data set. It helps to measure the capacity of the algorithm to support m times larger data set [19]. The sizeup measure is given by the following formula:

$$\text{Sizeup} = T_s/T_{ms} \tag{5.8}$$

    An algorithm that has a good sizeup takes m times longer from executing s and $m * s$.

- In order to evaluate the quality of the proposed method, the Running Time and the Quantization Error (QE) given by Eq. (5.5) are used. The QE measures the distance differences between a center and the data objects within that center.

Table 5.3 Comparison of the running time and quantization error of S-PSO versus existing methods

Data set	K-means	PSO	S-PSO
Magic	8	37	25
KDD	4350	10844	2741
House	500	2188	460
CoverType	134	1325	398
Magic	14693	10762	10861
KDD	1.3E12	7.1E11	6.8E11
House	114	53	49
CoverType	1002728	1053373	1061763

5.6.4 Comparison of the Performance of S-PSO Versus Existing Methods

In order to perform the experiments the following parameters are fixed, swarm size as 10 particles, the number of iterations as 50, inertia weight (w) as 0.72, acceleration coefficients $(c1, c2)$ as 1.49, and the number of clusters (k) as 5 for all the data sets except for magic and CoverType is respectively 2 and 7. Table 5.3 reports the performance results of the S-PSO compared to existing methods. Concerning the running time, S-PSO outperforms PSO for all the data sets and also outperforms k-means with house and kdd data sets but for magic and CoverType k-means seems to be faster. This could be explained by the fact that magic and CoverType are considered small data sets and they do not require to be clustered in a parallel manner. Instead, if they are processed in a parallel way it will cost additional time. Concerning the quantization error, PSO and S-PSO provide a better quality regarding k-means while S-PSO keeps almost the same quality as for PSO. This is due to the capacity of PSO and S-PSO using a population of candidate solutions in order to explore the search space.

5.6.5 Evaluation of the Impact of Time-To-Start Variable on the Performance of S-PSO

The purpose is to investigate the impact of the Time-To-Start variable on the performance of S-PSO. Table 5.4 outlines the results. When Time-To-Start variable increases and approaches the final stages of S-PSO, the running time gets more important but the quality gets better as the quantization error decreases. In fact, when the algorithm switches to k-means in its early stages which means when Time-To-Start is small, it takes advantage of the speed of K-means but it is deprived from the capacity of PSO of converging to high quality solutions. In fact the more the Time-To-Start value is small, the fastest the algorithm becomes, the less the quality is.

Table 5.4 Impact of Time-To-Start variable

	Time to start				
Data set	10	20	30	40	50
(a) The impact of Time-to-Start variable on the running time					
Magic	11	17	21	24	28
KDD	421	1416	2077	2789	3206
House	213	383	403	460	661
CoverType	114	229	260	398	493
(b) The impact of Time-to-Start variable on the QE values					
Magic	11301	10944	10630	10294	10365
KDD	9.1E11	9E11	8.9E11	8.9E11	8.9E11
House	65	58	58	50	53
CoverType	1294955	1150217	1139999	1101694	1192713

Therefore, it is important to figure out a compromise between time and quality. In our case, since PSO gets very slow near to convergence the Time-To-Start variable is chosen to be in the last ten iterations. This way switching to k-means will not affect the final result since S-PSO is almost converging and it will reduce the execution time.

5.6.6 Scalability Analysis

Figure 5.4a–d outlines the running time of the proposed method on D1M, D2M, D4M, and D8M, respectively. We can notice that for all data sets the running time decreases as the number of cores increases. For instance, for D4M, the running time decreases from 3987 with 1 core to 730 on 8 cores which means that it decreases over 5 times faster.

To evaluate the speedup for the proposed method, we maintain the constant data set size and we increase the number of cores from 1 to 8. Figure 5.5a–d illustrates the speedup results of respectively D1M, D2M, D4M, and D8M. The overall results show a good speedup for the proposed method. Actually, for all the data sets, when the number of cores goes from 1 to 4, the speedup results are very close to the linear and therefore a very good speedup. When the number of cores exceeds 4, S-PSO's performances start to decrease where the speedup is moving far from the linear. This is due to the additional communication time required to manage the increase of core's number.

Scaleup aims to measure the capacity of an algorithm to maintain the same running time while increasing the data set size with direct proportion to the number of cores. For evaluating the scaleup of our proposed method, we increase both the size of the data set and the number of cores. For investigating the scaleup, we used the D1M, D2M, D4M, and D8M data sets with respectively cores equal to 1, 2, 4, and 8. The results are plotted in Fig. 5.6. The ideal algorithm is the one having its

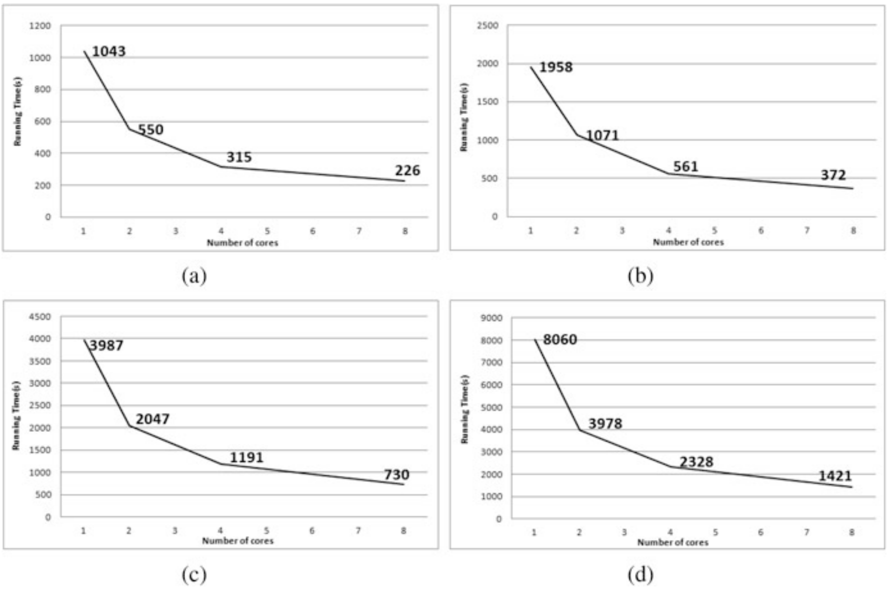

Fig. 5.4 Running time of S-PSO on the different simulated data sets. (**a**) D1M data set. (**b**) D2M data set. (**c**) D4M data set. (**d**) D8M data set

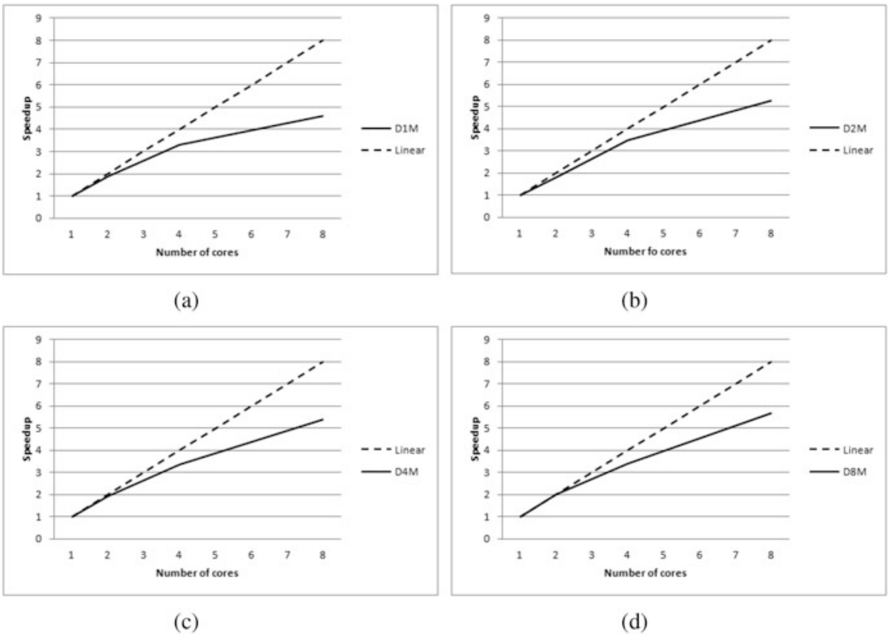

Fig. 5.5 Speedup results of S-PSO on simulated data sets. (**a**) D1M speedup. (**b**) D2M speedup. (**c**) D4M speedup. (**d**) D8M speedup

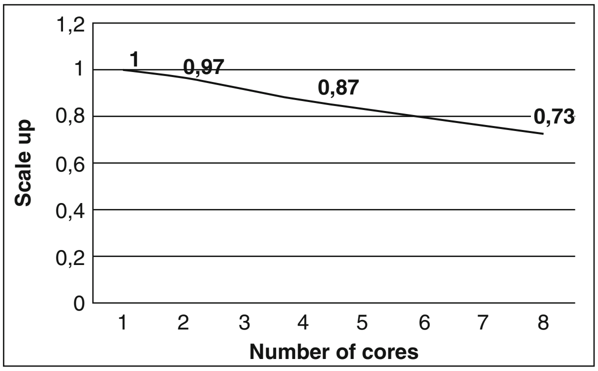

Fig. 5.6 Scaleup results of S-PSO

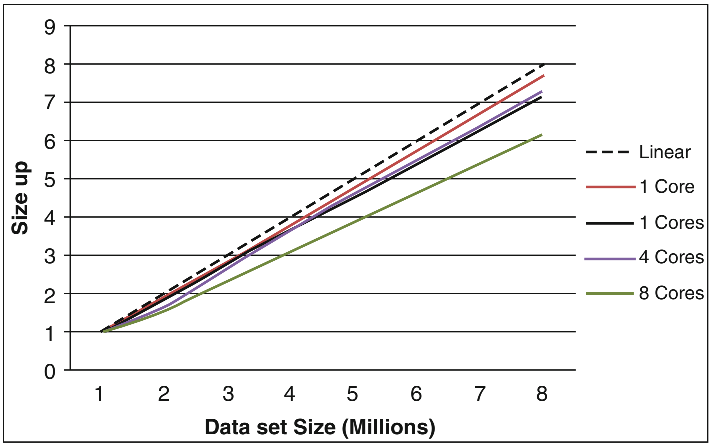

Fig. 5.7 Sizeup results of S-PSO

scaleup values very close or equal to 1. In our case, our proposed method shows a good scaleup. The overall results are approximately similar to D1M, D2M, D4M, and D8M using respectively 1, 2, 4, and 8 cores. Therefore S-PSO scales well with scaleup values ranging between 1 and 0.73.

To evaluate the sizeup of the proposed method S-PSO, the number of cores is kept constant while we increase the data set size in order to evaluate the behavior of S-PSO with the increasing volume of data. Figure 5.7 outlines the obtained sizeup results for respectively 1, 2, 4, and 8 cores. The obtained results show a good sizeup of our proposed method. For instance, for 1 core, the sizeup results are almost equal to the linear reaching 7.7 for D8M.

5.7 Conclusion

Big data clustering is an important field that requires special methods for dealing with high volume of data. Existing methods tried to fit the clustering algorithm based on PSO into MapReduce model. However, due to nonsuitability of MapReduce on iterative algorithms and the low convergence speed of PSO, we proposed a new large-scale data clustering method based on Spark framework and an adapted version of PSO combined with k-means. The proposed method, evaluated on both real and simulated data sets, has shown good results according to quality and scalability measures. As future works, we suggest combining S-PSO with other techniques to automatically looking for the number of clusters k since this parameter must be per-configured in advance. Moreover, we might think of applying feature selection algorithms to select most relevant features to use in S-PSO. This fact can reduce the heavy computation required in each iteration due to the high dimensionality.

References

1. A. Ahmadyfard, H. Modares, Combining PSO and k-means to enhance data clustering, in *International Symposium on Telecommunications, 2008* (2008), pp. 688–691
2. I. Aljarah, S.A. Ludwig, Parallel particle swarm optimization clustering algorithm based on MapReduce methodology, in *2012 Fourth World Congress on Nature and Biologically Inspired Computing (nabic)* (2012), pp. 104–111
3. G.P. Babu, M.N. Murty, Simulated annealing for selecting optimal initial seeds in the k-means algorithm. Indian J. Pure Appl. Math. **25**(1–2), 85–94 (1994)
4. M.A. Ben HajKacem, C.E. Ben N'cir, N. Essoussi, MapReduce-based k-prototypes clustering method for big data, in *Proceedings of Data Science and Advanced Analytics* (2015), pp. 1–7
5. M.E. Celebi, H.A. Kingravi, P.A. Vela, A comparative study of efficient initialization methods for the k-means clustering algorithm. Expert syst. Appl. **40**(1), 200–210 (2013)
6. C.P. Chen, C.-Y. Zhang, Data-intensive applications, challenges, techniques and technologies: a survey on big data. Inf. Sci. **275**, 314–347 (2014)
7. J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
8. K.-L. Du, M. Swamy, *Search and Optimization by Metaheuristics: Techniques and Algorithms Inspired by Nature* (Birkhäuser, Basel, 2016)
9. A.A.A. Esmin, D.L. Pereira, F. De Araujo, Study of different approach to clustering data by using the particle swarm optimization algorithm, in *IEEE Congress on Evolutionary Computation, 2008. CEC 2008 (IEEE World Congress on Computational Intelligence)* (2008), pp. 1817–1822
10. A.A. Esmin, R.A. Coelho, S. Matwin, A review on particle swarm optimization algorithm and its variants to clustering high-dimensional data. Artif. Intell. Rev. **44**(1), 23–45 (2015)
11. V. Gorodetsky, Big data: opportunities, challenges and solutions, in *Information and Communication Technologies in Education, Research, and Industrial Applications* (2014), pp. 3–22
12. K. Krishna, M.N. Murty, Genetic k-means algorithm. IEEE Trans. Syst. Man Cybern. B Cybern. **29**(3), 433–439 (1999)
13. S.A. Ludwig, MapReduce-based fuzzy c-means clustering algorithm: implementation and scalability. Int. J. Mach. Learn. Cybern. **6**(6), 923–934 (2015)

14. J. MacQueen et al., Some methods for classification and analysis of multivariate observations, in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1 (1967), pp. 281–297
15. R. Poli, J. Kennedy, T. Blackwell, Particle swarm optimization. Swarm Intell. **1**(1), 33–57 (2007)
16. R. Shyam, B.G. HB, S. Kumar, P. Poornachandran, K. Soman, Apache spark a big data analytics platform for smart grid. Proc. Technol. **21**, 171–178 (2015)
17. D. Van der Merwe, A.P. Engelbrecht, Data clustering using particle swarm optimization, in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03*, vol. 1 (2003), pp. 215–220
18. D. Xu, Y. Tian, A comprehensive survey of clustering algorithms. Ann. Data Sci. **2**(2), 165–193 (2015)
19. X. Xu, J. Jager, H.-P. Kriegel, A fast parallel clustering algorithm for large spatial databases, in *High Performance Data Mining* (Springer, Berlin, 1999), pp. 263–290
20. W. Zhao, H. Ma, Q. He, Parallel k-means clustering based on MapReduce, in *IEEE International Conference on Cloud Computing* (2009), pp. 674–679