Appl Intell
https://doi.org/10.1007/s10489-017-1089-7

Efficient data distribution and results merging for parallel data clustering in mapreduce environment

Abdelhak Bousbaci[1] · **Nadjet Kamel**[2]

Abstract Clustering data consists in partitioning it into clusters such that there is a strong similarity between data in the same cluster and a weak similarity between data in different clusters. With the significant increase in data volume, the clustering process becomes an expensive task in terms of computation. Therefore, several solutions have been proposed to overcome this issue using parallelism with the MapReduce paradigm. The proposed solutions in the literature aim to optimize the execution time while keeping the clustering quality close or identical to the sequential execution. One of the commonly used parallel clustering strategies when using the MapReduce framework consists in partitioning data and processing each partition separately. The results obtained from each partition are merged to obtain the final clusters configuration. Using a random data distribution strategy and an inappropriate merging technique will lead to an inaccurate final centroids and a rather average clustering quality. Hence, in this paper we propose a parallel scheme for partitional clustering algorithms based on MapReduce with a non-conventional data distribution and results merging strategies to improve the clustering quality. With this solution, in addition to optimizing the execution time, we exploit the parallel environment to enhance the clustering quality. The experimental results demonstrate the effectiveness and scalability of our solution in comparison with other recently proposed works. We also proposed an application of our approach to the community detection problem. The results demonstrate the ability of our approach to provide effective and relevant results.

Keywords Data clustering · Parallelism · MapReduce · Results merging · Data distribution · Genetic algorithm

1 Introduction

Clustering data consists in partitioning it into clusters such that there is a strong similarity between data in the same cluster and a weak similarity between data in different clusters. The main challenge of clustering is to find a compromise between clustering quality and execution time, and most works in the literature attempt to do just that.

Numerous clustering algorithms have been proposed using different techniques [13, 16, 18, 22, 30, 35]. With the significant increase in data volume, some clustering techniques become unsuitable to achieve the clustering task. For example, the hierarchical or the density-based clustering algorithms require an import execution time, due to their quadratic time complexity [25, 33]. Whereas, partitional clustering algorithms provide a reasonable execution time thanks to their linear complexity [30], with the exception of some cases like the K-medoids algorithm which has a quadratic complexity [36].

For these reasons, our study is based on partitional clustering algorithms.

The K-means algorithm [30] is the most common partitional clustering algorithm due to its simple implementation and effectiveness. Its complexity is $O(n * K * I * d)$ where

✉ Abdelhak Bousbaci
abousbaci@usthb.dz

Nadjet Kamel
nkamel@univ-setif.dz

[1] LRIA, Computer Science Department, USTHB, BP 32 El Alia 16111 Bab Ezzouar, Algiers, Algeria

[2] Computer Science Department, Faculty of Sciences, UFAS, Ferhat Abbas Setif University 1, Campus El Bez, Setif 1900, Algeria

Published online: 25 November 2017

n is the number of data points, K the number of clusters, I the number of iterations and d the data dimension.

Even with partitional clustering algorithms, clustering a large data amount requires an important execution time. To overcome this issue, several solutions have been proposed. Some works proposed to optimize the algorithm itself, whereas others opted for the use of parallelism. Parallelism can be applied using two major methods. The first method uses a network of connected machines [12, 20, 39] where the clustering algorithm is executed on a cluster of computers. The second method uses shared memory [23, 38](use of threads on a single multi-core processor).

Actually, applying a parallelism strategy for a clustering algorithm can have two different forms. The first one consists in parallelizing the algorithm tasks. To apply this strategy, the independent tasks are identified and executed simultaneously and independently. Hence, the said clustering algorithm has to be customized to fit this parallel form. For example in the K-means algorithm, the distance calculation and the centroids update are done by different processors or machines.

As this strategy follows the same behavior as the sequential algorithm, the resulted clustering quality is identical to that of the sequential execution. The only difference is the execution time.

In the second strategy, the processed data is divided into smaller data samples. Each one is processed independently with the given sequential clustering algorithm on a single processor or machine. With this strategy, we obtain intermediate results from each data partition, which are combined to obtain a global final result. This strategy can be applied with almost all partitional clustering algorithms. The parallel execution flow in this strategy differs from the sequential one. Hence, the generated clusters will be different from the ones obtained from a sequential execution. The clusters generated with this strategy are considered in most cases less accurate than the ones generated by the sequential execution. The difference in clustering quality is influenced by methods used in the data distribution and results merging phases. For the reasons cited above, most of the recent literature works on parallel clustering are based on the first parallelism strategy.

However, when applying the first strategy with iterative algorithms such as the K-means algorithm, the process will require as many MapReduce jobs as there are iterations in the sequential algorithm version. Indeed, the MapReduce model is not adapted for iterative algorithms as they require many MapReduce jobs, thus increasing the amount of data flow through the network, which in turn, will increase both the communication cost between the nodes and the execution time. Moreover, restarting MapReduce jobs on each iteration slows down the process. The second strategy, on the other hand, requires only one MapReduce job, which avoids the drawbacks of the first strategy. But, as cited above, the second strategy cannot provide a clustering quality equal to the sequential algorithm.

For these reasons, we propose a solution based on the second strategy to avoid the iterative MapReduce jobs. At the same time, we want to overcome the second strategy clustering quality issues.

Hence, our contribution is to propose a novel data flow model in MapReduce environment based on the second strategy using a non-conventional data distribution and results merging strategies. This will assure a good execution time and a clustering quality superior to the one obtained with the already cited strategies or with a sequential execution.

Thus, the improvement provided by our approach resides in the used data distribution and results merging methods. In this solution, our objective is to maximize the distance between the data point of the generated partitions, which is assured by the K-means algorithm. For the results merging process, we use the genetic meta-heuristic to exploit the results generated on each partition and avoid the irrelevant centroids. We illustrate our approach through the parallelization of the Sampling-PSO-Kmeans (SPKM) algorithm [21]. Our approach is tested and compared with recent works on parallel data clustering in the MapReduce environment. To investigate the effectiveness and the relevance of our proposed approach, we applied it to the community detection problem in Bibsonomy social bookmarking network. The conducted experimentations demonstrated the effectiveness and the scalability of our solution. Furthermore it outweighs the other tested methods in term of clustering quality.

The rest of the paper is organized as follows: Section 2 presents the related works. In Section 3, our proposed parallel clustering approach is detailed. Section 4 presents the implemented algorithms to validate our approach. Section 5 contains experimentations and obtained results. In Section 6, the application of our approach to the community detection problem is detailed. Finally, Section 7 presents the conclusion of our work and perspectives for future works.

2 Related work

Parallel clustering can be categorized into two major categories: parallel clustering using shared memory and parallel clustering using machines network.

2.1 Shared memory parallelism

Many works with different approaches have been proposed to parallelize the clustering process using shared memory [6, 23, 24, 38]. In [23], the authors proposed a solution

based on messaging between the processes. In [40] a parallel implementation for the fuzzy minimals clustering algorithm using shared memory has been proposed. Other works proposed to use multi-agent systems instead of simple threads as in [6].

In [8], a parallel clustering solution based on GPU is proposed. The authors presented a solution for parallelizing the flocking high dimensional data clustering problem (FHDC). The FHDC is an agent-based algorithm which means it is a computational intensive process. Hence, the solution was to parallelize it using GPU. The results showed that using a GPU provided an execution time 30 times faster than the one obtained using a CPU. GPUs provide a low cost/performance ratio.

Another solution based on GPU is proposed in [28]. The authors proposed to parallelize the Clustering Affinity Search Technique using GPU. The solution exploits the individual memory units of the GPU cores. The results demonstrated that by using a low performance GPU (250 Mb GPU memory) the algorithm performs 21 % faster than the sequential one.

In [42], the authors proposed a parallel documents clustering algorithm using a multi-core CPU. The proposed solution consists in hybridizing a parallel implementation of the K-means algorithm with a parallel version of the Principal Direction Divisive Partitioning (PDDP). The solution consists in using the PDDP algorithm to choose the initial centroids for the K-means algorithm. As both algorithms are computationally intensive when applied on large data sets, using parallelism was inevitable. The results proved once again that the parallel implementation surpasses the sequential one while keeping a good clustering quality.

In [24], the authors presented an approach based on the use of multi-cores processors and exploiting their cores to parallelize clustering algorithms. They parallelized K-means and K-nodes algorithms to cluster gene expressions. They opted for shared memory parallelism to avoid network communication and to be able to simulate a master-slave architecture. However, the disadvantage of this kind of parallelism lies in data access concurrency. To remedy this problem, using data locks is necessary, but at the same time it can create deadlock problems. The authors proposed McK-means, a parallelization of the K-means algorithm using shared memory. The proposed solution avoids deadlocks problems in addition to improving the execution time.

In McK-means algorithm, K-means is parallelized by simultaneously calculating the minimum distance and the centroids update.

Therefore, in this approach the initial data set is split into subsets, and the calculations of the minimum distance of the subsets are performed simultaneously by threads. On the other hand, each centroids is updated in parallel by an independent thread.

The most important parameter in this proposition is the choice of the number of threads. For the minimum distance search, the number of threads is equal to the number of available physical cores, and the number of threads for the centroids update phase is equal to K, where K is the number of clusters.

In this section several works presenting parallel solutions for clustering problems using CPU or GPU have been cited. The common point between them is the use of tasks parallelism. Only the algorithms independent tasks were parallelized and the datasets were treated entirely. This, will improve the CPU time compared to a sequential execution, whereas the clustering quality remains the same.

It has been demonstrated that a GPU has a better cost/performance ratio than a CPU due to the large number of cores contained in a GPU compared to a CPU, but not all computers are equipped with a dedicated GPU, whereas the CPU is a fundamental component in a computer. Hence, using solutions based on multi-core CPUs still have some advantages compared to the use of GPUs. Another solution for parallel clustering is the use of machines network. This method proved its efficiency as it combines the processing power of many machines to perform a single parallel process. In the next section, several parallel clustering algorithms will be presented and discussed.

2.2 Machines network parallelism

Parallel clustering using a network of machines is used generally to cluster large amounts of data. It consists in partitioning the data on the network nodes and processing each partition apart or distributing the clustering tasks among the machines of the network. Several methods were proposed for this, such as in [17] and [19]. The common factor between these works is that the authors proposed their own approach for the communication between machines and the definition of the parallelism, which requires advanced network programming skills. To avoid this, many authors, such as [12] and [43], opted for the MapReduce framework [11]. In this section we focus on the works based on MapReduce paradigm.

MapReduce is a software framework for processing very large amounts of data using a network of machines. In its simplest form, MapReduce is a two-phase process: a Map phase and a Reduce phase. In the Map phase a master node divides a problem into a number of independent parts that are assigned to Map tasks. Each Map task processes its part of the problem and outputs results as key-value pairs. The Reduce phase receives the outputs of the maps, where a particular reducer will receive only maps outputs with a particular key and process them. The power of MapReduce comes from the fact that Map and Reduce tasks can be distributed across different nodes. Therefore, MapReduce is

by design a distributed sorting platform. Since Map tasks are independent, they can be run in parallel similarly to Reduce tasks, which can be completed after the Map tasks are completed.

In [43], a parallel implementation of the K-means algorithm based on MapReduce was proposed. The solution consists in distributing the independents algorithm tasks among the machines of the network. In this work, the Map phase consists in calculating the minimal distance of each data point from the k centroids and assigning it to the closest one. In the Reduce phase the data points belonging to the same cluster are collected by a single reducer where the new centroid is calculated. The new generated centroids are sent again to the Mappers for a new iteration until the algorithm converges or the number of maximal number of iterations is reached.

In [12], the authors presented a solution for the K-means algorithm that differs from the one presented in [43]. This solution consists in dividing the initial dataset to many subsets, and each one is processed by a Reducer using the sequential K-means algorithm. The results generated from each Reducer are merged using the k-medoids algorithm on a single machine. Contrary to [43], this algorithm consists of one Map phase and one Reduce phase.

In [41], a MapReduce-based hybrid PSO K-means algorithm was proposed. The work uses the PSO to refine the initial centroids which are used by MapReduce based K-means as proposed in [43]. In [2], a parallel artificial bee colony algorithm for data clustering was proposed. The solution is based on the MapReduce paradigm, where only the computation of solutions fitness is parallelized. The results obtained demonstrated the efficiency of the solution and its scalability to large data sets.

Another parallel clustering solution is presented in [29]. This paper presented a parallel implementation of fuzzy C-means algorithm. The solution consists of two MapReduce jobs. The first one consists in computing the centroids matrix and the second one iterates over the data records and computes the distances between each record and the centroids. The results show that the proposed solution outperforms the sequential execution while keeping an identical clustering accuracy.

In [14], the authors used many connected machines to improve the execution of the K-means algorithm. They proposed ParC Algorithm. It consists in dividing the initial data on many connected machines and using the K-means algorithm on each one. At the end, the results obtained on each machine are merged together to get a final clusters configuration. The ParC algorithm can be summarized in three main phases:

1. Partitioning the initial Data on machines set.
2. Applying a clustering algorithm on each machine using its data partition to find the local clusters (β-clusters).
3. Merging the clusters found in the previous step that overlap together to get the final clusters configuration.

The difference between these propositions is that in some works like in [2, 41, 43] and in [29], the authors parallelized the algorithm tasks and used the entire dataset at once. In this category the parallelization improves the execution time and keeps a clustering quality identical to the sequential execution, whereas in the other works like in [12, 32, 37] or in [14], the processed data is divided randomly and each part is processed by a single machine, then the intermediate results are collected and merged on a single machine. With this approach, we improve the execution time, and, compared to the sequential execution, the clustering quality either increase or decrease according to the used data distribution and results merging strategies. All of the four cited works used a random data distribution.

For the results merging step, [32] used a simple merging strategy which consists in merging the centroids from the different partitions, such that each centroid from a given partition is merged with the *N-1* nearest centroids from each of the other partitions; where N is the number of all the partitions.

In [37] and [14], the authors used the overlap results merging strategy. This method consists in gathering the clusters that overlap in data space, according to a defined threshold. If we have two clusters A and B, and there are some data points from one of these two clusters that belong also to the other one, then these two clusters are merged together and constitute a new single cluster [5].

In the solution presented in [12], the authors used clustering results merging technique. This solution consists in clustering the centroids obtained from the different partitions and considering them as if they represent the entire data set. In this case, the K-medoids algorithm was used as it treated categorical data.

In the following, we present the algorithms we used to compare our approach (see Section 5.6).

2.2.1 Optimized big data K-means clustering using MapReduce (Opt MR-Kmeans)

Opt MR-Kmeans algorithm [9] is a parallel implementation of the K-means algorithm based on the MapReduce framework. It is a solution to avoid the K-means iterative processing dependency. In the classical MapReduce based K-means algorithm, each iteration is executed during a MapReduce job. This process requires a lot of network flow which will have negative impact on the overall execution time of the algorithm. To avoid this issue, the Opt MR-Kmeans algorithm consists in applying a sampling

phase on the processed data to reduce its size during a single MapReduce job. Then, the sampled data is clustered using the K-means algorithm during a second MapReduce job. In a final MapReduce job, the results obtained from the previous step are merged to obtain the final solution. All the processes require three MapReduce jobs, whereas the classical iterative MapReduce based K-means algorithm requires I MapReduce jobs; where I is the number of maximum iterations of the K-means algorithm. This algorithm greatly optimizes CPU time compared to the sequential K-means algorithm. As of the clustering quality, the results are identical or close to the ones resulted from the sequential K-means.

2.2.2 Parallel K-PSO based on MapReduce (Parallel KPSO)

In this work [41], the authors proposed a MapReduce based parallelization for the hybrid PSO-kmeans algorithm. The sequential PSO-Kmeans algorithm consists in refining the initial centroid for the K-means algorithm using the PSO algorithm. In this proposed parallel implementation, a sequential instance of the PSO algorithm is executed to refine the initial centroids. Next, the resulted centroids are used by a MapReduce based parallel K-means algorithm. The proposed parallel K-means algorithm consists in distributing the distance computing phase and centroids update phase on the network nodes. Data points are distributed on Mappers. Each Mapper calculates the distance between its data points and the centroids of the clusters and allocates each one to the closest cluster. In the Reduce phase, each reducer receives all the data points that belong to the same cluster and calculates its new centroid. The new centroids calculated by the reducers are used in a new MapReduce job until the maximum number of iterations is reached. This solution optimizes the execution time of the sequential hybrid PSO-kmeans algorithm. Contrary to the K-means phase, the PSO execution is not parallelized which will affect the overall algorithm performances. Moreover, the parallelization of the K-means algorithm is based on the iterative MapReduce K-means, where each iteration from the K-means algorithm is executed by a MapReduce job. This paradigm requires a lot of network communications and increases the connections cost.

3 Proposed approach

In this section, we present our proposed parallel clustering approach. First we present the global phases of the approach.

Then, before detailing each phase apart, we point out the differences between our solution and the literature works cited in Section 2. Finally, we go into the details of each phase of our approach.

The following three phases summarize the process of the parallel data clustering by partitioning we proposed.

- **S.1** Distributing data on the cluster nodes in the Map phase such that we maximize the distance between the data points of the different partitions (Map phase).
- **S.2** Applying clustering on each Reducer (Reduce phase).
- **S.3** Merging the intermediate results obtained from the Reducers on a single machine using the genetic algorithm.

As we can see, the first difference from the previous works is the method used to distribute data across the network machines. Almost all the proposed works neglected the data distribution step and used a random data distribution. On the contrary, we propose to use a non-random distribution. We propose to distribute data such as the similarity between the data instances of the different partition is low and between the data instances from the same partition is high. The second difference resides in the technique used to merge the intermediate results obtained from all the reducers. The literature works use merging techniques such as simple merging like in [32]. This simple merging method is not efficient, according to [9]. Using such merging techniques leads to inaccurate new centroids. If there are two centroids A and B to merge, which have respectively M and N data points in their clusters; with $M > N$, the new centroid will be closer to A cluster data points [5]. Other works used overlap technique like in [14] and [37]. This method showed its efficiency, but the obtained final number of centroids can differ from the number set initially (K). This can affect the clustering quality when the initial chosen number of clusters (K) is the optimal one. In [12], the merging strategy used consisted in clustering the intermediate results to generate the final solution. Beside the shortcomings cited above, all these merging techniques involve all the intermediate centroids including the irrelevant and inaccurate ones to generate the final solution. This will affect the performance of the algorithm.

After testing these techniques and analyzing each of the data partitions centroids (see Section 5), we noticed in some cases that the centroids resulted from a single partition give better results than the ones obtained after the merging process. This proved that the used merging technique failed to exploit the intermediate results and to obtain an optimal solution from the intermediate results. Hence, using an inappropriate merging technique provides inaccurate and malformed centroids.

From these observations, we propose to avoid using the standard fusion strategies and instead use a new method to merge the intermediate clustering results in a MapReduce

environment based on the genetic algorithm. This solution allows us to select only the most relevant centroids to constitute the final clusters configuration, which will improve the overall clustering quality.

In the following the proposed data distribution and results merging solution are detailed.

3.1 Data distribution module

In a parallel data clustering process, some data partitions can result in inaccurate and irrelevant clusters configuration, due to the data partitions creation method. These partitions affect the quality of the global clustering or in the best cases provide results identical or barely better than sequential execution. Part of this problem was solved using the genetic algorithm in the results merging step. To improve our last proposition, we suggest avoiding the random data distribution and propose a different method.

As data clustering aims to maximize the dissimilarity and the distance between data points of the different clusters, we propose to distribute data so as to increase the distance between the data instances of the different partitions. With this strategy, we expect that the centroids generated on each reducer will be as distant as possible from the centroids generated by the other Reducers with their own data partitions.

To achieve this, we propose to cluster data among the reducers (machines) instead of distributing it randomly. Each Reducer will process the data instances of a single cluster.

By clustering data among the machines, data points on each one will be most dissimilar to the other data points on the rest of the machines. By applying this data distribution strategy, the distance between the resulted centroids from the different partitions will also be maximized, which is the main goal of data clustering.

This solution consists in clustering the entire data set on the machines network. Although, this solution needs an important execution time. In [10], the authors proved that clustering a sample which represents 25% of the entire data set results in a pattern as precise as the one obtained from the entire data set. Therefore, to remedy the stand-off of the execution time, we propose to cluster only a sample from the data set. This step will allow us to create a clustering pattern which we use in the data distribution phase to classify data points of the entire dataset between the network machines. The K-means algorithm is used in this clustering step.

The proposed distribution process can be summarized in the following steps:

- Extracting a data sample representing 25% of the entire dataset.
- Clustering the data sample using the K-means algorithm with K equals to the number of machines (data partitions/Reducers).
- Sending the clusters pattern to the Mappers.
- During the Map phase, the Mappers compute the distance of each data sample and send it to the closest cluster (Reducer).

The Map (Data distribution phase) and Reduce phases are described in Algorithms 1 and 2 respectively.

Algorithm 1 Map phase

Input: C :Distribution centroids, Key: data instance index, Value: data instance
Output: (key,value) with key in the index of the Reducer and the value is the data instance.

```
instance:=Value;
Index:=1;
DistanceMin:=calculateDist(C1,Value);
for i ← 2 to C.length do
    d:=calculateDist(Ci,Value);
    if d < DistanceMin then
        DistanceMin:=d;
        Index:=i;
Key:=Index;
Value:=instance;
Output:(Key,Value) pair;
```

Algorithm 2 Reduce phase

Input: Key: Reducer index, Values: data instances belonging to the current reducer
Output: (key,values) with key is the index of the Reducer and the values are the intermediate centroids generated on the current Reducer.

```
instances[Values.length];
i:=1;
foreach value ∈ Values do
    instances[i]:=value;
    i++;
IntermediateCentroids:=Clustering(instances,K);
//Apply a local clustering on the local data partition
with K:number of clusters.
Value:=IntermediateCentroids;
Output:(Key,Value) pair;
```

By applying the K-means algorithm on the dataset, the distribution of data points on the machines may be unbalanced, resulting in partitions with a relatively small amount of data. In this case the data present on these partitions may not be sufficient to generate precise patterns. This can affect

the final clustering quality after the merging step if all the centroids are used in the final configuration. These observations drove us to avoid the standard merging strategies and propose the GA merging method.

3.2 Genetic algorithm based results merging module

In this section, we present our proposed solution for the results merging step [5].

As mentioned earlier, involving all the centroids may not be the optimal solution. Some centroids can be malformed due to a bad initialization in the clustering step (**S.2**), or due to the random data distribution on the machines.

We propose to select, from the set of centroids the K best ones. This can be seen as the K-means centroids initialization problem [26] which is an optimization problem. Therefore, we propose to use the genetic algorithm to select the best K centroids from the ones resulted from each data partition.

The Genetic Algorithm is a bio-inspired meta-heuristic used in optimization problems. The GA reproduces the behavior of the natural evolution system in order to generate solutions for a given optimization problem. By combining (crossover) and mutating solutions of a population, the GA generates new solutions from the original ones [31].

The merging step occurs after gathering the collection of results from the different partitions on a single machine.

In this solution, the genetic algorithm takes the centroids collection, from the clustering in step **S.2** on the different data partitions, as input data for the generation of the population.

To optimize the evaluation step of the generated chromosomes (potential solutions) from the population, only a sample from the initial data set is used so that the evaluation step will not require an important execution time. In a final step, the best generated chromosome will be applied on the entire data set to obtain the final fitness value representative of the solution.

3.2.1 Population and chromosomes

The genetic algorithm takes as input data the centroids set $\bar{C}$ where $\bar{C} = \bigcup_i^n C_i$, with c_i the centroids set from the i^{th} data partition. The initial population of the genetic algorithm is generated randomly from the centroids set $\bar{C}$. N chromosome are generated, and at each iteration the best N ones are selected. Each chromosome from the population is a vector of K random centroids selected from the centroids set. Figure 1 represents a potential solution in the case where the number of clusters k is equal to 6, and C_{25} is the 5^{th} centroids from the second data partition and so on. Each chromosome is a potential clusters solution.

Fig. 1 Chromosome solution

3.2.2 Crossover and mutation operations

At each iteration of the genetic algorithm, new chromosomes are obtained by applying mutation and crossover operations on the population. The crossover operation combines two randomly selected solutions and generates two new solutions as shown in Fig. 2. The mutation operation in our case consists in changing randomly one of the K elements of a given chromosome and replacing it by another from the set of centroids. The best solutions are selected for the next iteration and so on until the last iteration. At the end of the genetic algorithm, the best obtained solution represents the final clusters configuration.

The entire process of our proposition can be illustrated by Fig. 3.

In the following, we study the complexity of our proposal.

3.3 Algorithm complexity

In this section we detail and discuss the complexity of our proposed solution. Our solution consists in three main steps:

1. Data distribution using K-means.
2. Clustering data partitions using SPKM.
3. Merging intermediate results using GA.

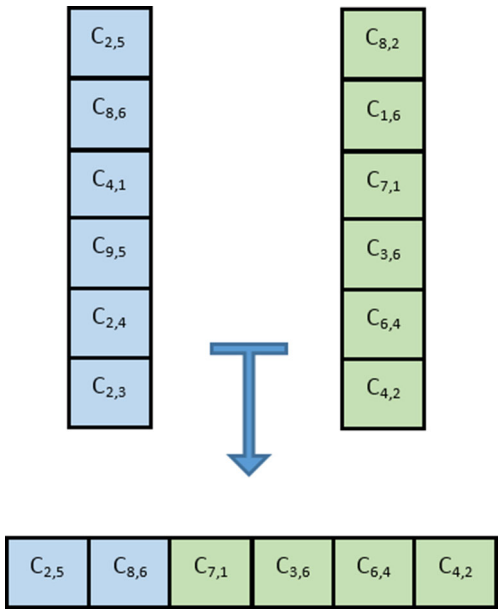
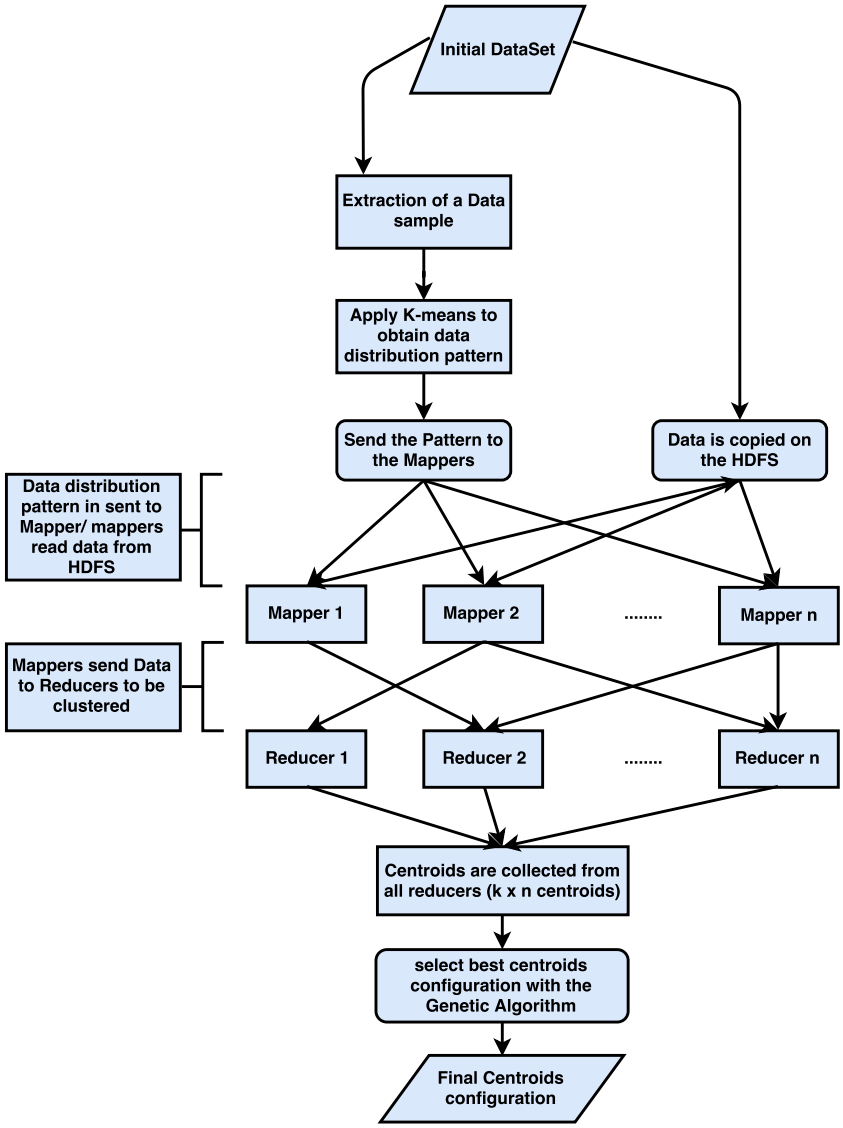


Fig. 2 Chromosomes crossover

Fig. 3 Parallel architecture



Let n be the size of the processed dataset, d the dimension of the data instances (number of attributes), and k the number of clusters. With $n >> k$ and $n >> d$. In the following the complexity of each phase of the proposed parallel scheme is presented.

3.3.1 Data distribution phase complexity

In the first phase, the initial data set is divided into N partitions using a clustering pattern, where N is the number of the used nodes. The pattern is created using the K-means algorithm with a data sample from the initial data set. The time complexity of this step is $O(k' * m * n' * d)$; with k' the number of clusters which is equal to the number of nodes in this case; m: is the number of iterations; n': is the size of the data sample ($n' << n$).

After generating the pattern, we use it to distribute the data points of the initial dataset among the network nodes. Therefore, the distance between each data point and the centroids of the generated pattern is calculated. The complexity of this step is $O(k' * n * d)$; where k': is the number of clusters (nodes); n: is the size of the initial data set; and d is the data dimension. The data distribution process is executed in parallel by the Mappers. Hence, the complexity becomes $O(k' * n * d/nb\_Nodes)$. As the number of clusters (k') in this step is equal to the number of nodes, the complexity will be $O(k' * n * d/k') = O(n * d)$. The entire first step complexity is equal to $O(k' * m * n' * d) + O(n * d)$.

Assuming that $k' * m * n'$ is larger than n the complexity becomes $O(k' * m * n' * d)$.

3.3.2 Clustering phase complexity

The second phase takes most of the execution time of the entire solution as it represents the core of the clustering process. As our proposed parallel scheme can be implemented with different partitional clustering algorithms, the complexity of this phase depends on the used clustering algorithm. Therefore, in this analysis we present the complexity of the SPKmMR algorithm in order to study the complexity of the entire proposed scheme. The SPKmMR algorithm consists of three phases: a) A sampling phase using the K-means algorithm; b) Initial centroids refinement using the PSO algorithm; c) A final clustering phase using the K-means algorithm with the centroids obtained from the execution of the PSO algorithm.

The complexity of the sampling phase (a) is $O(k * m * n * d)$, which is the same as the K-means algorithm , with k: the number of clusters; m: the number of iterations; n: the dataset size and d: number of attributes.

In the PSO algorithm phase (b), the most time-consuming operations are the population generation and the solutions evaluation which use the fitness formula (1). Therefore, the PSO complexity is defined by the complexities of these operations. The population generation complexity is $O(d * k * p)$, with d the number of attributes and k the number of clusters($d * k$ represents the dimension of the problem) and p the size of the population. The fitness function assigns each data point to the closest centroid and calculates the average distance between each data point and its centroids. This process is applied for each particle from the swarm until the maximum number of iterations is reached, so the complexity of this phase is defined by $O(k * n * d * p * i)$ where p is the number of particles and i is the maximum number iterations. The complexity of the PSO algorithm will then be $O(d * k * p) + O(k * n * d * p * i)$. The population generation complexity will be neglected compared to the particles evaluation, therefore the PSO algorithm complexity will be $O(k * n * d * p * i)$.

The final step is a K-means execution using the centroids resulted from the PSO algorithm. As mentioned earlier the K-means complexity is $O(k * m * d * n)$. The global complexity of the clustering step is the sum of the sampling,PSO and K-means algorithms complexities which is: $O(k*m*n*d) + O(k*n*d*p*i) + O(k*m*d*n) -> O(k*n*d*p*i) + O(k*m*d*n) -> O(k*n*d*p*i)$.

All the clustering process is parallelized using MapReduce, thus the complexity becomes: $O(k * n * d * p * i/Nb\_Nodes)$. From this complexity we can notice that the time complexity decreases linearly with the increasing of the number of nodes which is illustrated in Figs. 4a, 5a, 6a and 7a.

3.3.3 GA merging phase complexity

In this phase the genetic algorithm is used to select the final clusters configuration. Similarly to the PSO algorithm, the GA complexity is represented by the complexity of the population generation and the fitness evaluation. The population generation complexity is $O(d * k * p)$, with d is the number of attributes and k the number of clusters and p is the number of chromosomes in the population.

The fitness function is the same used with the PSO algorithm, hence its complexity is $O(k * n'' * d * p * g)$, where k: the number of clusters; n'': the size of the used data sample to evaluate the solutions; d: number of attributes; p: size of the population and g: the number of generations. So the complexity of the results merging phase is $O(d * k * p) + O(k * n'' * d * p * g)$. The time complexity of the population generation is neglected because it is much smaller than the time complexity of the solutions evaluation. Therefore, the time complexity of the GA algorithm becomes $O(k * n'' * d * p * g)$.

The global complexity of the parallel scheme is the sum of the 3 presented phases: $O(k' * m * n' * d) + O(k * n * d * p * i/Nb\_Nodes) + O(k * n'' * d * p * g)$. Assuming that $k' << k$ and $n'' << n' << n$, the complexities of data distribution and results merging phases become negligible. The global complexity is represented then by $O(k * n * d * p * i/Nb\_Nodes)$.

4 Implementation

To test and validate our proposition, we implemented two parallel clustering algorithms. the first is the parallel K-means using MapReduce (KmeansMR), and the second is our proposed algorithm in [4] the SPKmMR algorithm. In the following we details the SPKmMR algorithm.

In [4], we proposed a parallelization of the SPKM algorithm [21] which we called Parallel Sampling-PSO-Multi-Core-K-means using MapReduce (SPKmMR). The SPKM algorithm is based on hybrid PSO-K-means algorithm [7] and the sampling process. It improves the results by sampling the initial data set before using PSO algorithm. It divides data into S subsets and applies K-means on each subset. After that, each subset is represented by only its centroids. This step will reduce the global amount of data by keeping only the most representative data points. This process improves the execution time and the efficiency of the

PSO algorithm, which represents an upswing of the entire approach.

This algorithm can be summarized by the following four steps:

1. S sub-samples are selected from the initial data set.
2. K-means is applied on each sub-sample.
3. The resulted centroids from precedent step are used with PSO algorithm considering them as swarm particles (*S* particles).
4. K-means is applied on the entire data set using the initial configuration obtained from the PSO algorithm.

We use numerous machines to parallelize SPKM algorithm, and exploit each of the CPUs cores by locally parallelizing SPKM algorithm.

We describe this process by the following three steps:

1. Distributing data randomly on the cluster nodes in the Map phase.
2. Applying sampling-PSO-MultiCoreK-means on each Reducer.
3. Merging results obtained from the Reducers on a single machine using the simple merging.

The PSO algorithm is sensitive to large datasets. With the sampling method presented in [21], the size of the data set is reduced and PSO can reach an optimal solution with a smaller number of iterations, which means a shorter execution time. Therefore, in our approach, we proposed to reduce the initial data set by splitting it to several machines before starting to apply SPKM algorithm. In a final step, the results are merged with the simple method.

Sequential processes are executed on many machines in parallel. Since the used CPUs have many cores, using a sequential program on them does not exploit all their potential. So, we use local parallelism instead of the global network parallelism. In Sample-PSO-K-means, K-means algorithm is used twice. First, it is used in the sample step, and then in the final clustering step. Thus the local parallelism will focus on K-means. This parallelism is done through the following steps:

1. Data instances are separated on P partitions according to the number of available physical cores, and a thread is created for each part (P threads).
2. Each of the P threads is charged to compute the minimal distance of its partition.
3. Other K threads are created and attributed to the K cluster; K is equal to the number of centroids.
4. Each of the K threads is responsible of computing the centroid of its own cluster after each update.

The entire process of the algorithm can be defined as follows:

1. Initial data is divided on the set of machines in the Map phase, and each data partition is sent to a particular reducer.
2. On each Reducer(machine), the following steps are applied using its own data partition:
   (a) Data is divided again into S sub-partitions.
   (b) A sampling algorithm is applied on each data partition using McK-means.
   (c) Results from sampling algorithm are used as initial data for the PSO algorithm.
   (d) Apply McK-means algorithm using the results obtained from PSO algorithm as initial centroids.
3. The Reducers sent their results (intermediate centroids) to a single machine, where clusters of each configuration are merged using a simple merging process to get the final configuration.

5 Experiments

In this section, the different approaches and steps of our proposal are experimented and evaluated.

First, the implemented algorithms, SPKmMR and the KmeansMR, are tested with different solutions for data distribution and results merging including our proposed solution. Through these experimentations, we study and analyze the influence of these techniques in term of clustering quality on the parallel data clustering in MapReduce environment.

Secondly, we experiment the performances of the SPKmMR with our proposed parallelism scheme in terms of CPU time. Thirdly, we analyze and discuss our algorithm convergence.

Finally, we compare our global proposition which is the SPKmMR using the no-random data distribution and the GA merging with recently proposed parallel clustering algorithms.

5.1 Experiments setup

For the experimentations, we used a cluster of 16 machines. Each node is equipped with a Dual Core CPU (2 physical cores) and 2 GB of RAM. The cluster runs on Linux Ubuntu 10.04. We used the Hadoop 1.2.1 framework which is an open source implementation of the MapReduce framework.

The experiments were applied on a synthetic dataset (DataSet1) [15] and a real dataset "The Individual household electric power consumption"(DataSet2) [27] which represents the electrical consumption. The dataset represents the measurements of electric power consumption in household over a period of almost 4 years.

To test the scalability of our approach and study its behavior with different datasets sizes, we generated two other synthetic data sets DataSet3 and DataSet4 from DataSet1 and DataSet2, respectively, by duplicating them multiple times. Table 1 describes the four data sets used.

5.2 Solution evaluation

Many metrics have been proposed to evaluate and validate a clustering solution. Through the realized experiments we used two clustering validation metrics. The first metric determines how compact the clusters are. It calculates the average distance between each centroid and the data points belonging to its cluster. We minimize this value for a better clustering. We use formula (1) to calculate this metric which represents the average sum of squares within cluster (SSW).

$$SSW = \frac{\sum_{i=1}^{k} \left\{ \frac{\sum_{j=1}^{n_i} d(o_i, p_{ij})}{n_i} \right\}}{K} \tag{1}$$

where: p_{ij} is the j^{th} data point in the i^{th} cluster; o_i is the center of the i^{th} cluster; $d(o_i, p_{ij})$ is the distance between the data point p_{ij} and the centroid o_i; n_i is the number of data instances in the cluster C_i and K is number of clusters. The Euclidean distance is calculated with formula (2).

$$d(i_i, i_j) = \sqrt{\sum_{m=1}^{i_a} (i_{im} - i_{jm})^2} \tag{2}$$

where i_i and i_j are two vector instances; i_a the size of an instance; i_{im} and i_{jm} are the m^th attributes of the two instances.

We note that formula (1) is used to calculate the fitness for the PSO algorithm particles as well as the GA solutions (chromosomes) fitness.

The second metrics we used is the Davies−Bouldin index (DBI), which represents a ratio between intra-clusters distance and inter-clusters distance. The lower the DBI value the better the clustering solution. It can be calculated with formula (3)

$$DBI = \frac{1}{K} \sum_{i=1}^{K} R_i \tag{3}$$

Table 1 DataSets description

DataSets	Number of records	Dimensions
DataSet1	164860	3
DataSet2	2075259	9
DataSet3	989160	3
DataSet4	10376295	9

where R_i is the maximum ratio between the cluster i and the other k clusters with $i \neq j$. R_{ij} is calculated using formula (4) where S_i and S_j represent respectively the average distance between the points within cluster i and j as calculated in formula (1). d_{ij} is the distance between the centroids of the cluster i and j.

$$R_{ij} = \frac{S_i - S_j}{d_{ij}} \tag{4}$$

5.3 Data distribution and results merging experiments (Clustering quality)

To evaluate the data distribution and results merging processes, we compared them with the standard random data distribution and simple merging and clustering merging (K-means) techniques. All these techniques are applied on the SPKmMR and the KmeansMR algorithms (Section 4) to demonstrate the improvement provided by our proposal to different partitional parallel clustering algorithms. In the following we conduct 3 different experiments. The first concerns the baseline algorithm. The second is about the proposed GA results merging technique compared to the standard techniques. In the third and last experiment, we demonstrate the influence of the proposed data distribution technique in a MapReduce environment.

5.3.1 Experiment 1: baseline algorithms

In this first experiment we analyze the sequential versions of the parallel algorithms SPKmMR and KmeansMR which are the SPKM algorithm and the standard K-means algorithm. The experiments are conducted on DataSet1 and DataSet2.

Tables 2 and 3 represent the parameters set for the PSO and the K-means algorithms used in SPKM and K-means algorithm. These parameters are used for all the next implemented algorithms.

Table 4 shows the results obtained after the execution of the K-means algorithm, the SPKM and its parallel version SPKmMR using simple results merging on both DataSet1 and DataSet2.

Table 2 PSO algorithm parameters

Parameter	Value	
	DataSet1	DataSet2
Inertia factor	0.5	0.3
Confident coefficient at its best position	0.1	0.72
Confident coefficient at its neighboring	3.0	1.49

Table 3 K-means parameters

Parameters	Value
Number of Data partitions (for MapReduce implementation)	16 (number of machines)
Number of clusters	11 (DataSet 1-3) / 50–100 (DataSet 2-4)
Number of iterations in K-means	15 (DataSet 1-3) / 25 (DataSet 2-4)
Number of threads(for Multi-core K-means)	2 (dynamic according to the used CPU)

Results discussion The results presented in Table 4 show that with DataSet2 the SPKM and the SPKmMR algorithms give better fitness values than the K-means algorithm does. This is due to the use of PSO and sampling for selecting the initial centroids. We can note also that the SPKmMR algorithm improves the fitness value comparing to the SPKM algorithm. In fact, dividing the data on the network nodes reduces the dataset size on each machine. Therefore, the PSO algorithm produces an optimal solution with the same number of iterations to the one obtained when processing the entire dataset.

On the DataSet1, we can notice that the SPKM algorithm gives improvement compared to the K-means algorithm. Unlike the results obtained on the DataSet2, we can see that SPKM algorithm provides a slightly better results than its parallel implementation (SPKmMR). This unexpected result is due to the results merging strategy that we used (See next section).

Indeed, the improvement obtained with the parallel implementation of the SPKM algorithm on both DataSets can be improved by changing the results merging strategy. In the following we discuss the results obtained after applying our proposed results fusion strategy.

5.3.2 Experiment 2: results merging comparison

In this second experiment, we analyze the influence of the merging step in parallel clustering and the improvement provided by the proposed results merging technique using GA. To evaluate our approach we compared it with simple merging and K-means merging techniques. All of the three implemented merging techniques: Simple merging, K-means merging and GA merging are used with our proposed SPKmMR algorithm and the KmeansMR algorithm.

Table 4 Clustering quality (SSW)

Algorithms	Data sets		
	DataSet1	DataSet2 (k = 50)	DataSet2 (k = 100)
K-means	0.079604	0.079233	0.067021
SPKM	0.077303	0.073251	0.063955

Table 5 Genetic algorithm parameters

Parameter	Value	
	DataSet1	DataSet2
Population size	100	200
Number of generations	100	50
Crossover %	100%	100%
Mutation %	80 %	80%

Table 5 represents the parameters used in the GA in the merging step.

Table 6 presents the results obtained by the SPKmMR and KmeansMR algorithms using the simple, K-means and GA merging techniques on DataSet1 and DataSet2.

Results discussion From Table 6, we can notice that the K-means merging technique provides better results than the simple merging technique. If we compare the fitness obtained by the sequential SPKM on DataSet2 when $k = 100$ from Table 4 to the fitness obtained by SPKmMR where $k = 100$ in Table 6, we note that the difference between the fitness values is not important. The same results can be seen for DataSet1 in Tables 4 and 6. This is due to the inability of the simple merging to exploit the results obtained from the different data partitions. Subsequently, we notice that the GA merging strategy outperforms the simple and the K-means merging in all cases with both of the DataSets used. This is because this technique selects only the best K centroids, from the centroids set obtained from the different data partitions, and it avoids the malformed ones.

If we compare the results obtained from the KmeansMR and the SPKmMR algorithms, on Table 6 for DataSet2, we can notice that improvements obtained by using SPKmMR algorithm over KmeansMR are more important when we use the GA merging than when we use the K-means or the simple merging. For example, when we use GA merging with K = 100, SPKmMR fitness is equal to 0.049195 and KmeansMR fitness is equal to 0.051913. The difference of fitness values is more important than it is when we use K-means merging, where SPKmMR fitness is equal to 0.059569 and KmeansMR fitness is equal to 0.061751. This case demonstrates that GA merging exploits the results generated by SPKM on each of the data partitions.

5.3.3 Experiment 3: data distribution results

In this final step, we test our approach for improving the data distribution process in a MapReduce environment.

Table 6 Clustering quality with the different merging strategies

Merging techniques	KmeansMR			SPKmMR		
	DataSet1	DataSet2 (k = 50)	DataSet2 (k = 100)	DataSet1	DataSet2 (k = 50)	DataSet2 (k = 100)
Simple merging	0.079888	0.077204	0.064242	0.0775832	0.071459	0.063574
Kmeans merging	0.0788292	0.071763	0.061751	0.077153	0.070317	0.059569
GA merging	0.077211	0.057541	0.051913	0.0767487	0.054363	0.049195

To evaluate the data distribution process, we implemented the KmeansMR, the SPKmMR algorithms and the results merging techniques discussed in the previous section (simple, K-means and GA merging). As the used MapReduce cluster is composed of 16 machines, we clustered the dataset sample (% 25 of the initial dataset) on K = 16 clusters for both used DataSets using the K-means algorithm. The obtained clusters patterns are used with the two implemented parallel clustering algorithms (KmeansMR and SPKmMR) to distribute data on the machines network.

Table 7 represents the results obtained by KmeansMR and SPKmMR algorithms using the Simple, K-means and GA merging after applying our data distribution strategy with both DataSets.

Results discussion By analyzing these results, we note that, in all cases, the fitness values obtained with the simple and K-means merging are inaccurate, with the different clustering algorithm and the different DataSets. If we compare the results from Table 7 when using the simple and the K-means merging with the results from Table 4 for DataSet2, we can see that the fitness values returned by the sequential SPKM algorithm outperform those obtained by the parallel algorithms KmeansMR and SPKmMR while using the simple or the K-means merging. Furthermore, the sequential K-means algorithm provides better results. We notice the same remark for DataSet1. These results are due to the merging technique used and not to the data distribution process.

In the following (Table 8), we analyze the intermediate results obtained with the centroids configuration from each data partition of the used DataSets while using the SPKmMR and KmeansMR algorithms. For DataSet1 we present the intermediate results of SPKmMR algorithm, and for DataSet2 we present intermediate results of KmeansMR in the case when $K = 100$.

From Table 8, we notice that some partitions from DataSet2 provides better results than the ones obtained after the merging process (using simple or K-means merging), like the partitions $P3$, $P4$, $P11$.. etc., whereas, some partitions provide bad clusters configuration, like in $P1$, $P2$, $P5$, .. etc. That is because, in some cases when applying our proposed data distribution method, some partitions will have a relatively small size. This is insufficient to generate a correct clustering pattern.

Contrary to the previous cases, on Table 7 we can see that when we use the GA merging we obtain the best results. By choosing and merging the best clusters configuration from the different data partitions, we obtained the best fitness value with the SPKmMR algorithm while using the GA merging. An important point to highlight, is that the KmeansMR algorithm provides better results than the sequential SPKM algorithm (see Table 4 DataSet2). The KmeansMR algorithm also outperformed the SPKmMR algorithm when applied after a random data distribution with all the merging strategies. From all the results we can see that the improvement obtained in DataSet2 is more important that the one obtained in DataSet1. This is due to the datasets size. DataSet1 is relatively smaller than DataSet2, so the improvement after applying the partitioning was less important than with DataSet2, because the clustering algorithms can reach a sufficient result without the data size reduction benefit provided by the parallel implementation.

Table 7 Results after applying our data distribution with different merging strategies

Merging techniques	KmeansMR			SPKmMR		
	DataSet1	DataSet2 (k = 50)	DataSet2 (k = 100)	DataSet1	DataSet2 (k = 50)	DataSet2 (k = 100)
Simple merging	0.159748	0.270103	0.277858	0.156695	0.3009458	0.2812698
Kmeans merging	0.101498	0.094805	0.080865	0.097269	0.092251	0.079740
GA merging	0.076442	0.045391	0.037061	0.075649	0.043310	0.034448

Table 8 Intermediate results from SPKmMR

Data partitions	DataSet1	DataSet2
P1	0.159275	0.134110
P2	0.136808	0.111946
P3	0.123890	0.077897
P4	0.149114	0.043421
P5	0.123619	0.129709
P6	0.141003	0.074752
P7	0.134218	0.076125
P8	0.146532	0.057816
P9	0.146205	0.185012
P10	0.121721	0.070572
P11	0.150959	0.062136
P12	0.176330	0.122667
P13	0.155793	0.245486
P14	0.129691	0.105169
P15	0.155635	0.054617
P16	0.134052	0.125209

All these remarks prove the importance of data distribution and results merging steps in parallel data clustering.

5.4 CPU time and SpeedUp experiments

In this section we analyze and discuss the results in term of CPU time, SpeedUp and efficiency.

We conducted different experiments to study the behavior of our proposition. The first experiment is a comparison in term of CPU time between the sequential and the parallel execution. The second is a study of the comportment of the SPKmMR algorithm when executed through our proposed parallel scheme, with a variation in the number of involved nodes.

5.4.1 Experiment 1: comparison of sequential and parallel execution(CPU Time)

We compare the CPU time of the parallel execution (SPKmMR and KmeansMR) with the sequential execution (SPKM and K-means).

The following results represent the execution time results from DataSet2 when k = 100 with the different implemented algorithms. Table 9 represents the execution time of the sequential algorithms compared with the parallel algorithms while using our proposed parallel clustering scheme (non-random data distribution + GA results merging technique). From Table 9 we can see that the two implemented parallel algorithms surpass theirs sequential versions in term of CPU time. We can see that KmeansMR is more than 61% faster than its sequential version, and the SPKmMR is more than 72 % faster than the SPkm.

Table 9 CPU time

Algorithms	Execution time (minutes)
Sequential Kmeans	18.46
Sequential SPKM	28.07
KmeansMR	7.11
SPKmMR	7.61

5.4.2 Experiment 2: CPU time and SpeedUP with variation in number of nodes

In the following, we study the performance of our approach in term of execution time and speedup by varying the number of the involved nodes (2, 4, 8 and 16). The experiments are conducted with the SPKmMR algorithm which provided the best clustering quality.

The speedup is a metric which determines how much a parallel algorithm is faster than a sequential one, or how much a parallel algorithm using x nodes is faster than the same algorithm using y nodes where $x > y$. SpeedUp can be calculated with formula (5)

$$SpeedUp = \frac{T_y}{T_x} \tag{5}$$

where T_y is the CPU time using y nodes and T_x is the CPU time using x nodes.

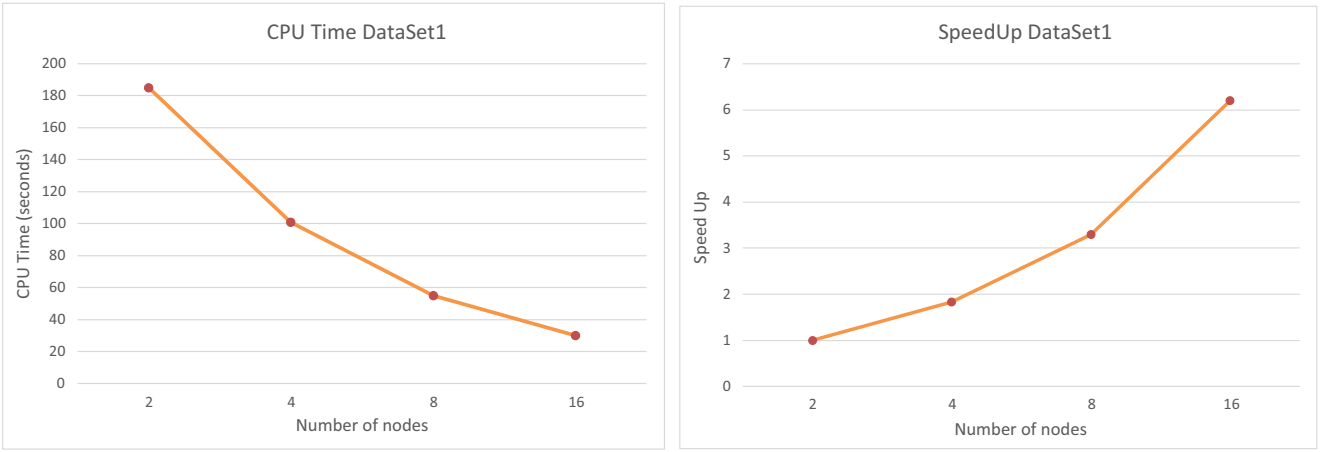
The tests are conducted on the 4 datasets cited in Table 1.

We note that, in these experiments, the execution time of the merging process (using GA) is not counted because it is executed on a single machine and takes nearly the same amount of time with all the datasets. We also note that for the speedup computation, we considered that the baseline case is when we use 2 nodes and not when we use the sequential algorithm, because the sequential and the parallel algorithms do not present the same execution scheme. So for the case using 2 nodes the speedup will be equal to 1.

Figures 4, 5, 6 and 7 represent the CPU time and SpeedUp graphs for the 4 datasets.

We can notice that with all the datasets the CPU time decreases linearly and at the same time the SpeedUp increases linearly.

We observe that the SpeedUp value is more important on DataSet3 compared to DataSet1, and also on DataSet4 compared to DataSet2. Thus, with a larger dataset the SpeedUp increases. But if we compare the results from DataSet1 with the results from DataSet2 we see that speedup with DataSet1 is more important than with DataSet2 despite the larger size of DataSet2 (the same between DataSet3 and DataSet4). This unexpected observation is due to the used non-random data distribution. With this method the generated data partitions will have different sizes contrary



(a) CPU Time. (b) SpeedUp.

Fig. 4 CPU time and SpeedUp with number of nodes variation on DataSet1

to the random data distribution where all partitions will have mostly the same size. In fact, with our proposed data distribution approach, the SpeedUp will depend on the used dataset. With DataSet1, the smallest generated partition contains 4072 rows and the largest contains 20445, which represent approximately 5 times the size of smallest partition.

Therefore, when using 16 machines, for example, the global execution of the clustering process will be represented by the time taken by the machine processing the largest data partition. This will affect the SpeedUp performance of the algorithm.

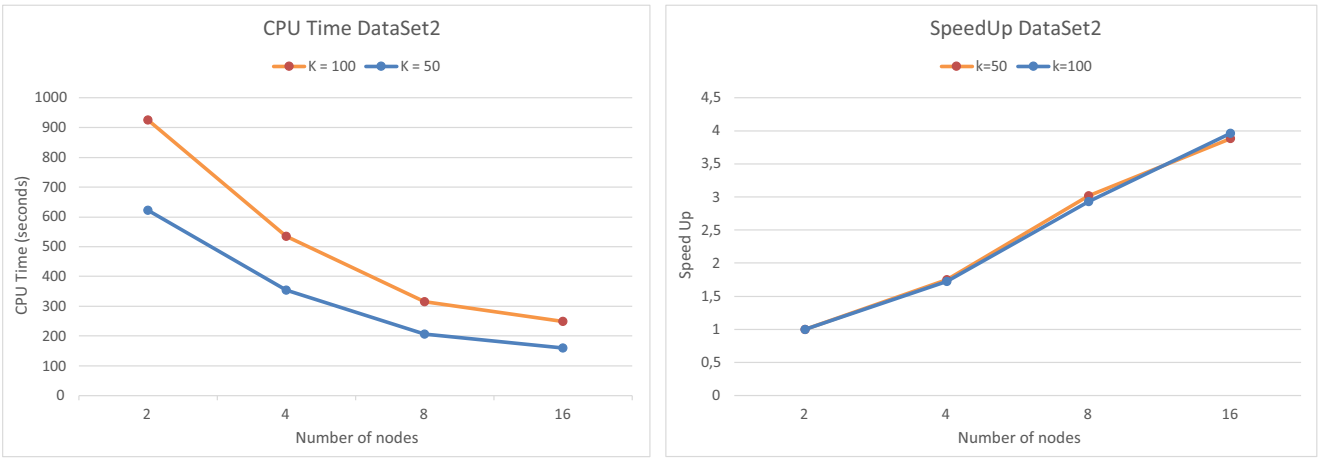
For DataSet2, the largest data partition (280809 rows) is 10 times larger than the smallest data partition (27933), which will affect the speedup performances more than it will on DataSet1.

To clearly visualize this case we compute the efficiency of our approach with the different datasets. The efficiency of a parallel algorithm is the ratio of the speedup to the number of used nodes. We calculate the efficiency using Formula (6)

$$Efficiency = \frac{SpeedUp}{Nb\_nodes} \tag{6}$$

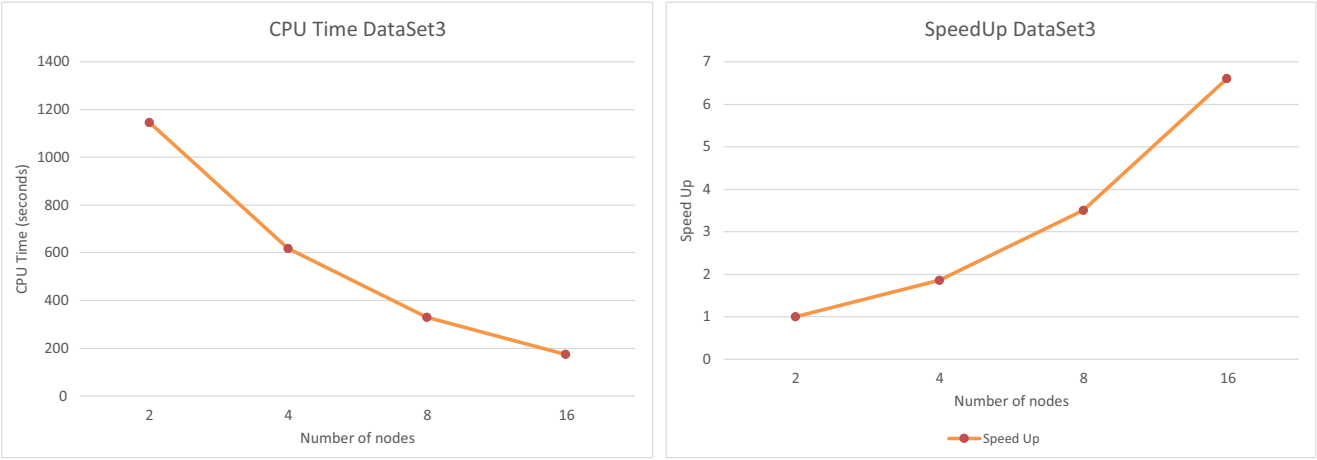
Table 10 contains the efficiency values obtained from the different datasets using the SPKmMR algorithm with our proposed parallelism scheme.

From Table 10, we can notice that the efficiency is improved as the data size increases. This can be noticed clearly with DataSet3 which is a duplication of DataSet1.



(a) CPU Time. (b) SpeedUp.

Fig. 5 CPU time and SpeedUp with number of nodes variation on DataSet2



(a) CPU Time. (b) SpeedUp.

Fig. 6 CPU time and SpeedUp with number of nodes variation on DataSet3

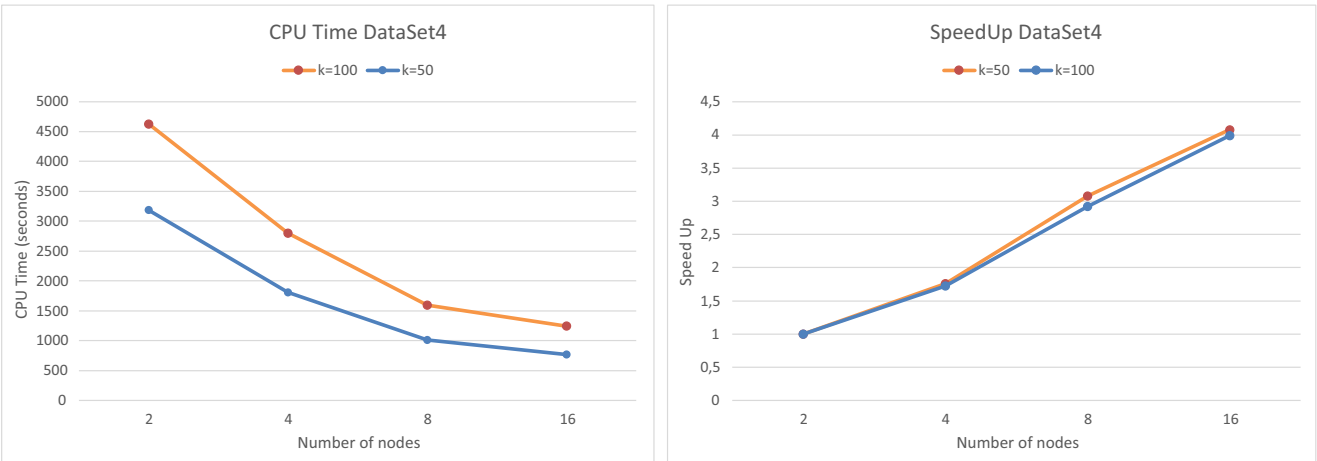
A slight improvement is obtained with DataSet4 compared to DataSet2 which is due to the same problem cited above (data partitions size).

5.5 Convergence analysis

In this section we analyze and discuss the convergence of our proposed solution while used with the SPKM algorithm. We analyze the convergence of the entire parallel scheme and the local convergence of the different data partitions. We also compare these measurements with the convergence of the sequential SPKM algorithm using the entire data set. The convergence results while using the GA merging at a given iteration, represents the final result obtained by merging the intermediate results obtained from the different partitions at that iteration.

The experiments were conducted on DataSet1 and DataSet2 (case where k = 50). Figure 8, represents the convergence graph on DataSet1.

As most of the data partitions present almost the same convergence behavior, we chose to present only 3 partitions: the smallest partition P4 with 4072 rows, the largest one P5 with 20445 rows and a third random data partition P9 with 12359 rows. From the graph we can notice that the sequential algorithm which uses the entire dataset converges until reaching the 15^{th}iteration, whereas the partitions converge



(a) CPU Time. (b) SpeedUp.

Fig. 7 CPU time and SpeedUp with number of nodes variation on DataSet4

Table 10 Approach efficiency on different datasets

DataSets	Efficiency
DataSet1	0.770
DataSet2	k = 50 0.485 / K = 100 0.496
DataSet3	0.823
DataSet4	k = 50 0.510 / K = 100 0.499

almost within the 10^{th} iteration. The smallest partition P4 converges within 5 iterations due to its small number of rows. The partition P5 and P9 which have a greater number of rows than P4 take 10 iterations to converge. We also notice that the final results obtained from the different partitions after being merged using the genetic algorithm converge at the 5^{th} iteration.

Figure 9, illustrates the convergence graph on DataSet2. As for DataSet1, we present only 3 partitions convergence. Partition P6 with 48575 rows, partition P10 with 172927 rows and partition P15 with 280809 rows. From Fig. 9 we notice that the sequential execution of the SPKM algorithm takes 25 iterations to converge as it processes the entire dataset at once. Contrary to the sequential execution, we notice that the partition P6 reaches a stable state after 5 iterations due to its small number of data points. As partitions P10 and P15 have a greater number of data points, they both converge after the 15^{th} iteration. We also notice that the final results obtained after the merging using GA reach a stable state after the 15^{th} iteration. Besides the improvement obtained in term of execution time by processing the data in parallel, we can notice that by dividing the initial dataset into smaller partitions, the number of iteration required by each one of those partitions to converge is fewer than the iterations required to process the entire dataset. This also improves the execution time.

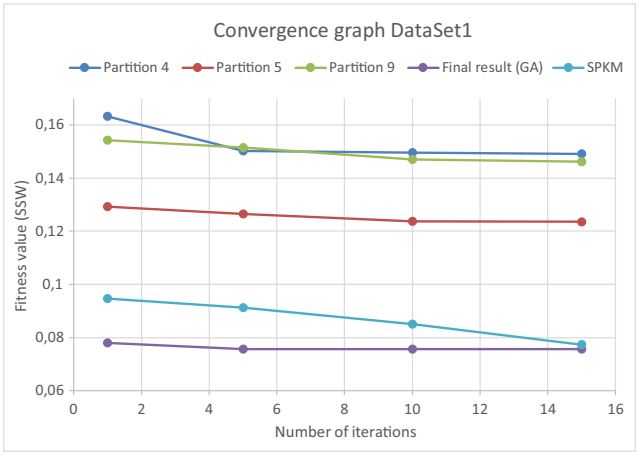


Fig. 8 Convergence on DataSet1

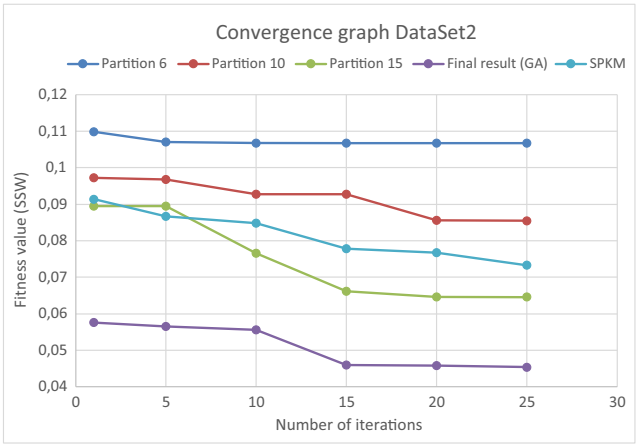


Fig. 9 Convergence on DataSet2

5.6 Algorithms comparison

In this last section of the experiments, we compare our parallel approach with some recently proposed parallel clustering algorithms. The conducted experiments concern the clustering quality and the algorithms parallelism efficiency.

We compared our parallel clustering scheme on the SPKmMR algorithm with the standard sequential K-means, a MapReduce-based hybrid PSO K-means algorithm "Parallel K-PSO based on MapReduce" (Parallel KPSO) [41] and an optimized MapReduce K-means algorithm "Optimized big data K-means clustering using MapReduce" (Opt MR-Kmeans) [9].

The evaluations were conducted on all the 4 datasets shown in Table 1.

Tables 11 and 12, represent respectively the parameters of the implemented $ParallelKPSO$ and $OptMR-Kmeans$ algorithms.

In a first test we evaluated the clustering validity using the SSW and the DBI metrics.

Table 13, summarizes the results obtained in term of clustering quality.

From Table 13, we notice that our approach once again provides the best results among all the algorithms on all the datasets. The difference in clustering quality between DataSet1 and DataSet3 (the same for DataSet2 and DataSet4) is minimal, which demonstrates the scalability of the algorithm to different size problems.

In a final experiment, we compare the efficiency of our approach with the other algorithms.

Table 11 Parallel KPSO parameters

Parametre	Values	
	DataSet1/DataSet3	DataSet2/DataSet4
Inertia factor	0.5	0.3
Confidence Best position	0.1	0.72
Confidence neighbor	3.0	1.49
Number of cluster	11	50
Number of iterations (K-means)	15	25
Number of nodes	16	16

The following results are obtained from DataSet4, which does not present the best efficiency according to Table 10. We chose to conduct the tests on DataSet4 to avoid the best case scenario and analyze our approach in its worst case.

From the results in Table 14, we can see that our approach presents an acceptable and competitive efficiency compared to the other algorithms.

6 Application to the community detection problem

To demonstrate the effectiveness and relevance of the results obtained with our approach, we applied it to the community detection problem in the Bibsonomy social bookmarking network [3]. The community detection problem consists in discovering a set of users who share common points of interest on a social network. Data clustering is one of the solutions used to solve this problem, whereby each cluster contains a potential set of similar users. In [34], the authors proposed a solution based on the k-means in order algorithm to detect users' communities in the academic social bookmarking system Bibsonomy.

To do so, the similarity between the different users is calculated and used to create a distance matrix. Each element of the matrix represents the distance between two users from the social network.

Table 12 Opt MR-Kmeans parameters

Parameters	Values	
	DataSet1/DataSet3	DataSet2/DataSet4
Number of clusters	11	50
Number of iterations	15	25
Number of nodes	16	16

The similarity between two different users is calculated using formula (7) [34].

$$Sim\left(User_i, User_i\right) = \left|tags_{Ui} \cap tags_{Uj}\right| / \left|tags_{Ui} \cup tags_{Uj}\right| \quad (7)$$

Where $tags_{Ui}$ and $tags_{Uj}$ represent the tags used by $user_i$ and $user_j$ respectively, to annotate their bookmarks. The distance matrix is created using formula (8) [34].

$$Mat\_dist\,[i,\,j] = \begin{cases} 1/Sim\left(User_i, User_j\right), & if\ Sim\left(User_i, User_j\right) > 0 \\ 100000, & if\ Sim\left(User_i, User_j\right) = 0 \\ 0, & if\ i = j \end{cases} \quad (8)$$

Where $Mat\_dist[i,\,j]$ is the distance between $user_i$ and $user_j$. Therefore, each row i from the matrix represents the distance values between $user_i$ and all the other users of the collection. The distance matrix will be used as input data for the used clustering algorithm. This will allow us to find the users' communities.

6.1 Experiments

In this section, we present the details of the experiments realized to apply our approach on the community detection problem in Bibsonomy.

6.1.1 Bibsonomy dataset

To apply our approach on the community detection in Bibsonomy we used the "2016-01-01" version of the Bibsonomy dataset [1]. The dataset contains 3 different files: tas,bookmarks and bibtex files. In our experiments we only use the "tas" file which contains the tags used by each user to annotate his own bookmarks. The information contained in the "tas" file will be used to calculate the similarity

Table 13 Clustering validation

	K-means		Parallel KPSO		Opt MR Kmeans		Our Approach	
Validation metrics	DBI	SSW	DBI	SSW	DBI	SSW	DBI	SSW
DataSet1	1.035278	0.079604	0.970420	0.077609	1.030236	0.079442	0.940197	0.075649
DataSet2	1.19255	0.079233	1.165538	0.074396	1.204050	0.080650	1.057580	0.043310
DataSet3	1.031842	0.079399	1.017055	0.077940	1.022694	0.079817	0.946913	0.075928
DataSet4	1.196346	0.080747	1.169648	0.075900	1.192127	0.080011	1.065280	0.04571

between the users and subsequently create the distance matrix. We used a sample of 1078 users from the dataset, which resulted in a $1078x1078$ distance matrix.

6.1.2 Solution evaluation

To evaluate the results obtained with our approach, we use the average square of within clusters (SSW) metric using formula (1). To examine the relevance of the generated results, we use a human evaluation on the generated clusters. The human evaluation consists in analyzing the generated clusters and determining which users are actually similar according to theirs used tags. With the human evaluation, we can verify the pertinence between the users of each cluster. When we compare two clustering solutions, the better one should have a higher number of similar users [34].

6.1.3 Experimentations and results

We implemented the solution for this problem with the SPKmMR algorithm and compared it with the k-means, the Opt-Kmeans and the Parallel KPSO algorithms. We conducted numerous tests on the distance matrix using the K-means algorithm to determine the number of clusters (K). We chose two different values for K: 10 and 20. The first experiment consists in applying the implemented algorithm on the generated distance matrix and evaluating the results using the SSW metric. Each generated cluster represents a potential community.

Table 15 represents the results obtained in term of clustering quality using the SSW metric.

From Table 15, we note that the Opt-Kmeans provides results almost similar to the ones obtained with the k-means algorithm. The Parallel KPSO provides results slightly superior to those obtained with the k-means and the Opt-kmeans algorithms. We note that our approach provides the best results with the two different values of K.

In the second experiment, we aim to confirm the superiority of our approach seen in the first experiment. To do this, we use the human evaluation to analyze and compare the clusters generated with our approach to the ones generated with the k-means and the Parallel KPSO algorithms. We analyze the case where $K = 10$. With the human evaluation we can determine the real number of similar users inside each cluster. A set of users are considered similar if they have multiple common tags. To realize this, we use the "tas" file which contains the tags of each user and compare them.

Table 16 represents the obtained results in term of the number of similar users on each cluster using the K-means algorithm, the Parallel KPSO algorithm and our approach.

From the results presented on Table 16, we can distinguish again the quality and the superiority of the results provided by our approach. The total number of similar users found using our approach is more important than the ones generated by the k-means and the Parallel KPSO algorithms, which proves the effectiveness of our solution. The human evaluation confirmed the results provided by the clustering analysis using the SSW metric.

From these results we can conclude that our solution provide effective and relevant results for real clustering problems.

Table 14 Algorithms efficiency on DataSet4

Algorithms	Efficiency
Parallel KPSO	0.6
Opt MR Kmeans	0.521
Our approach	0.510

Table 15 Clustering quality (SSW)

Algorithms	K = 10	K = 20
K-means	6.8506	6.6333
Opt-Kmeans	6.8963	6.6205
Parallel KPSO	6.5692	6.3461
Our approach	5.8317	5.2788

Table 16 Number of similar users on each cluster

Algorithms	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Total
K-means	10	12	14	37	34	7	61	18	101	9	303
Parallel KPSO	45	9	8	16	41	51	101	20	58	17	357
Our approach	5	26	63	20	52	31	16	77	66	30	386

7 Conclusion

In this paper we proposed a parallel scheme for partitional clustering algorithms based on MapReduce environment.

The objective of our approach is to improve data distribution on clusters nodes and exploit the centroids obtained from each one using a GA based results merging strategy.

The proposed approach is validated and tested on two parallel clustering algorithms, SPKmMR and KmeansMR algorithms.

Using our parallel scheme, the two algorithms provide improvement in clustering quality compared to when executed with conventional data distribution and results merging strategies.

The SPKmMR algorithm using our parallel scheme has been tested with four data sets of different sizes to test the scalability of our approach. Furthermore, it was compared with recent parallel clustering algorithms on four datasets.

The experimental results demonstrated that the SPKmMR algorithm using our parallel scheme outweighed the $OptMR - Kmeans$ [9] and the $ParallelKPSO$ [41] algorithms in clustering quality and provided a competitive parallelism efficiency in its worst case.

In a final experiment, our approach was applied on the community detection problem in Bibsonomy and compared it with the k-means algorithms and the Parallel KPSO algorithms. Once again, the results demonstrated the effectiveness of our solution.

All the conducted experiments proved the effectiveness of our proposal and the importance of the data distribution and results merging processes in parallel data clustering. As future work, we aim to improve our data distribution strategy using meta-heuristics.

References

1. (2016) Knowledge and Data Engineering Group, University of Kassel: Benchmark folksonomy data from bibsonomy version of January 01st. http://bibsonomy.org/
2. Banharnsakun A (2017) A mapreduce-based artificial bee colony for large-scale data clustering. Pattern Recogn Lett 93:78–84
3. Benz D, Hotho A, Jäschke R, Krause B, Mitzlaff F, Schmitz C, Stumme G (2010) The social bookmark and publication management system bibsonomy. The VLDB Journal—The International Journal on Very Large Data Bases 19(6):849–875
4. Bousbaci A, Kamel N (2014) A parallel sampling-pso-multi-core-k-means algorithm using mapreduce. In: 14th international conference on hybrid intelligent systems (HIS), 2014. IEEE, pp 129–134
5. Bousbaci A, Kamel N (2016) Efficient results merging for parallel data clustering using mapreduce. In: 13th international conference distributed computing and artificial intelligence. Springer, pp 349–357
6. Chaimontree S, Atkinson K, Coenen F (2011) A multi-agent based approach to clustering: harnessing the power of agents. In: International workshop on agents and data mining interaction. Springer, pp 16–29
7. Cui X, Potok TE (2005) Document clustering analysis based on hybrid pso+ k-means algorithm. J Comput Sci (special issue) 27:33
8. Cui X, Charles JS, Potok T (2013) Gpu enhanced parallel computing for large scale data clustering. Futur Gener Comput Syst 29(7):1736–1741
9. Cui X, Zhu P, Yang X, Li K, Ji C (2014) Optimized big data k-means clustering using mapreduce. J Supercomput 70(3):1249–1259
10. Davidson I, Satyanarayana A (2003) Speeding up k-means clustering by bootstrap averaging. In: IEEE data mining workshop on clustering large data sets
11. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
12. Ene A, Im S, Moseley B (2011) Fast clustering using mapreduce. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 681–689
13. Ester M, Kriegel HP, Sander J, Xu X et al. (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd, vol 96, pp 226-231
14. Ferreira Cordeiro RL, Traina Junior C, Machado Traina AJ, López J, Kang U, Faloutsos C (2011) Clustering very large multi-dimensional datasets with mapreduce. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 690–698
15. Fränti P (2015) Clustering datasets. http://cs.uef.fi/sipu/datasets/
16. Goil S, Nagesh H, Choudhary A (1999) Mafia: efficient and scalable subspace clustering for very large data sets. In: Proceedings of the 5th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 443–452
17. Guerrieri A, Montresor A (2012) Ds-means: distributed data stream clustering. In: European conference on parallel processing. Springer, pp 260–271
18. Guha S, Rastogi R, Shim K (1998) Cure: an efficient clustering algorithm for large databases. In: ACM SIGMOD Record, ACM, vol 27, pp 73–84
19. Hammouda KM, Kamel MS (2014) Models of distributed data clustering in peer-to-peer environments. Knowl Inf Syst 38(2):303–329
20. Han D, Giraud-Carrier C, Li S (2015) Efficient mining of high-speed uncertain data streams. Appl Intell 43(4):773–785
21. Kamel N, Ouchen I, Baali K (2014) A sampling-pso-k-means algorithm for document clustering. In: Genetic and evolutionary computing. Springer, pp 45–54

22. Kaufman L, Rousseeuw PJ (2009) Finding groups in data: an introduction to cluster analysis, vol 344. Wiley
23. Kerdprasop K, Kerdprasop N (2010) A lightweight method to parallel k-means clustering. International Journal of Mathematics and Computers in Simulation 4(4):144–153
24. Kraus JM, Kestler HA (2010) A highly efficient multi-core algorithm for clustering extremely large datasets. BMC Bioinforma 11(1):1
25. Kriegel HP, Kröger P, Sander J, Zimek A (2011) Density-based clustering. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1(3):231–240
26. Kwedlo W, Iwanowicz P (2010) Using genetic algorithm for selection of initial cluster centers for the k-means method. In: International conference on artificial intelligence and soft computing. Springer, pp 165–172
27. Lichman M (2013) UCI machine learning repository. http://archive.ics.uci.edu/ml
28. Lin KW, Lin CH, Hsiao CY (2014) A parallel and scalable cast-based clustering algorithm on gpu. Soft Comput 18(3):539–547
29. Ludwig SA (2015) Mapreduce-based fuzzy c-means clustering algorithm: implementation and scalability. Int J Mach Learn Cybern 6(6):923–934
30. MacQueen J et al. (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth berkeley symposium on mathematical statistics and probability, Oakland, CA, USA, vol 1, pp 281–297
31. Maulik U, Bandyopadhyay S (2000) Genetic algorithm-based clustering technique. Pattern Recogn 33(9):1455–1465
32. More P, Hall LO (2004) Scalable clustering: a distributed approach. In: IEEE international conference on fuzzy systems, 2004. Proceedings. 2004, IEEE, vol 1, pp 143–148
33. Rokach L, Maimon O (2005) Clustering methods. In: Data mining and knowledge discovery handbook. Springer, pp 321–352
34. Saoud Z, Platoš J (2017) Community detection in bibsonomy using data clustering. In: International conference on information systems architecture and technology. Springer, pp 149–158
35. Sheikholeslami G, Chatterjee S, Zhang A (1998) Wavecluster: a multi-resolution clustering approach for very large spatial databases. In: VLDB, vol 98, pp 428–439
36. Shirkhorshidi AS, Aghabozorgi S, Wah TY, Herawan T (2014) Big data clustering: a review. In: International conference on computational science and its applications. Springer, pp 707–720
37. Sinha A, Jana PK (2016) A novel k-means based clustering algorithm for big data. In: International conference on advances in computing, communications and informatics (ICACCI), 2016. IEEE, pp 1875–1879
38. Stoffel K, Belkoniene A (1999) Parallel k/h-means clustering for large data sets. In: European conference on parallel processing. Springer, pp 1451–1454
39. Sun Z (2013) A parallel clustering method study based on mapreduce. In: 1st international workshop on cloud computing and information security. Atlantis Press
40. Timón I, Soto J, Pérez-Sánchez H, Cecilia JM (2016) Parallel implementation of fuzzy minimals clustering algorithm. Expert Syst Appl 48:35–41
41. Wang J, Yuan D, Jiang M (2012) Parallel k-pso based on mapreduce. In: IEEE 14th international conference on communication technology (ICCT), 2012. IEEE, pp 1203–1208
42. Xu S, Zhang J (2004) A parallel hybrid web document clustering algorithm and its performance study. J Supercomput 30(2):117–131
43. Zhao W, Ma H, He Q (2009) Parallel k-means clustering based on mapreduce. In: IEEE international conference on cloud computing. Springer, pp 674–679

Abdelhak Bousbaci is a PhD. student in computer science at the University of Sciences and Technology Houari Boumediene (USTHB), Algeria, since 2012. He received his Master degree in Intelligent Computer Systems from the USTHB in 2012. His research interests are related to Data Mining, Data Clustering and Parallel Data Clustering.

Nadjet Kamel is a full professor at the department of computer science of the University Ferhat Abbes of Setif 1 (UFAS1), Algeria, since 2011. She received her Magister and the Phd degree in Computer Science from the University of Science and Technology Houari Boumediene (USTHB), Algeria, respectively in 1995 and 2007. She has been a Postdoctoral Researcher at the University of Moncton, in Canada from August 2007 to August 2009 and a Lecturer at the USTHB from 1995 to 2007. From 2009 to 2011 she was an associate professor at the same university. She has been involved in many research projects (PNR and CNEPRU). Since 2011, she is the head of the team research "Data Mining and Machine Learning" at the Laboratory of Research in Artificial Intelligence (LRIA) at USTHB. She has been the Head of the department of computer science of UFAS1 from 2014 to 2015. Since 2015, she is the Vice President of the UFAS1. Her main interests are related to Computational intelligence and Data Mining. She organized and served as program commitee member of many international conferences.