Chapter 1
Overview of Scalable Partitional Methods for Big Data Clustering

Mohamed Aymen Ben HajKacem, Chiheb-Eddine Ben N'Cir, and Nadia Essoussi

1.1 Introduction

Clustering, also known as cluster analysis, has become an important technique in machine learning used to discover the natural grouping of the observed data. Often, a clear distinction is made between learning problems that are supervised, also known as classification, and those that are unsupervised, known as clustering [24]. The first deals with only labeled data while the latter deals with only unlabeled data [16]. In many real applications, there is a large supply of unlabeled data but limited labeled data. This fact makes clustering more difficult and more challenging than classification. Consequently, there is a growing interest in a hybrid setting, called semi-supervised learning [11] where the labels of only small portion of the observed data are available.

During the last four decades, many clustering methods were designed based on different approaches such as hierarchical, partitional, probabilistic, and density-based [24]. Among them, Partitional clustering methods have been widely used in several real-life applications given their simplicity and their competitive computational complexity. This category of methods aims to divide the dataset into a number of groups based on the optimization of one, or several objective criteria. The optimized criteria may emphasize a local or a global structure of the data and its optimization is based on an exact or an approximate optimization technique. Despite the competitiveness of the computational complexity of partitional methods compared to other methods, it fails to perform clustering on huge amounts of data

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

C.-E. Ben N'Cir (✉)
University of Jeddah, Jeddah, KSA

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_1

[22]. In fact, given the exponential growth and availability of data collected from different sources, analyzing and organizing these data has become an important challenge referred to as *Big data Analytics*. This challenge has been the focus of several types of researches which proposed scalable partitional clustering methods based on different acceleration techniques. The aim of this work is to give a theoretical and empirical overview of scalable big data clustering methods. We propose a new categorization of these methods based on the used methodology to improve scalability. For each defined category, we review the existing methods and we describe the used acceleration technique or framework. Also, an empirical evaluation is given using most representative methods from each of the defined categories using large simulated and real datasets.

The remainder of the chapter is organized as follows. Section 1.2 presents partitional clustering methods. Then, Sect. 1.3 reviews the existing scalable Big data partitional clustering methods. After that, Sect. 1.4 gives an experimental evaluation of Big data partitional clustering methods on different simulated and real large datasets. Finally, Sect. 1.5 presents conclusions and some perspective points.

1.2 Partitional Clustering Methods

More than thousands of clustering methods were proposed in the literature. Commonly used methods are usually classified according to the fundamental concept on which clustering methods are based. This classification leads to defining the following categories [24]: hierarchical, partitional, probabilistic, and density-based. Among them, partitional clustering methods remain the most commonly used because of their simplicity and their competitive computational complexity.

Partitional clustering methods try to organize data into k clusters (where k is an input parameter), by optimizing a certain objective function that captures a local and global structure of grouping. Most of the partitioning methods start with an initial assignment and then use an iterative relocation procedure which moves data points from one cluster to another to optimize the objective function. Examples of these methods are k-means [36], k-modes [26], k-prototypes [25], and fuzzy c-means [7]. The k-means [36] is the most fundamental partitional clustering method which is based on the idea that a center can represent a cluster. After selecting k initial cluster centers, each data point is assigned to a cluster based on a Euclidean distance measure, then k cluster centers are recomputed. This step is repeated until an optimal set of k clusters are obtained based on an objective function. The k-modes [26] uses the simple matching coefficient measure to deal with categorical attributes. The k-prototypes [25] integrates k-means with k-modes methods to partition mixed data. On the other hand, partitional clustering methods often generate partitions where each data point belongs to one and only one cluster. The fuzzy c-means [7] extends this notion to associate each data point a membership function to each cluster.

Despite the efficiency of partitional clustering methods, they do not scale with a huge volume of data [47]. This is explained by the high computational cost to build grouping when dealing with a large amount data. To overcome this weakness, several works were proposed to improve the efficiency of conventional partitional clustering methods. This survey emphasizes on scalable Big data partitional clustering methods, specifically those extending k-means, k-prototypes, and fuzzy c-means methods. We present in the next section an overview of these methods.

1.3 Big Data Partitional Clustering Methods

To deal with large-scale data, several methods were designed in the literature which are based on coupling conventional partitional clustering methods and acceleration techniques. These methods aim to enhance the speed of clustering process by reducing the computational complexity. We propose in this work to classify these methods based on the acceleration technique used to improve the scalability. Four categories are defined, *parallel methods*, *data reduction-based methods*, *centers reduction-based methods*, and *hybrid methods*. Figure 1.1 shows a classification tree of these methods where the depth of the tree represents the progression in time and the width of the tree represents the different categories and subcategories. We detail the following the main characteristics of each category.

1.3.1 Parallel Methods

Parallelization is one of the most used acceleration techniques, which aims to reduce the computational cost of conventional partitional clustering methods. Parallelization is defined as a process where the computation is divided into parallel tasks. Several parallel partitional methods were proposed in the literature. These methods are motivated by the assumption that distance computations between one data point with cluster centers are independent of those between any other data point and cluster centers. Thus, distance computation between different data points and cluster centers can be executed in parallel. The parallelization can be done using different frameworks such as Message Passing Interface (MPI) [46], Graphics Processing Unit (GPU) [38], MapReduce [15], or Spark [50]. In the following, we present an overview of parallel clustering methods which are based on these frameworks.

1.3.1.1 MPI-Based Methods

MPI is a parallel framework designed to process a large amount of data through cluster of machines. It is deployed in master/slave architecture where the master sends tasks to slaves and receives computed results. While slaves receive tasks, process them, and send results to the master. MPI provides set of functions which are used by developers to create parallel applications. These functions aim to communicate and exchange data and messages between machines. For example,

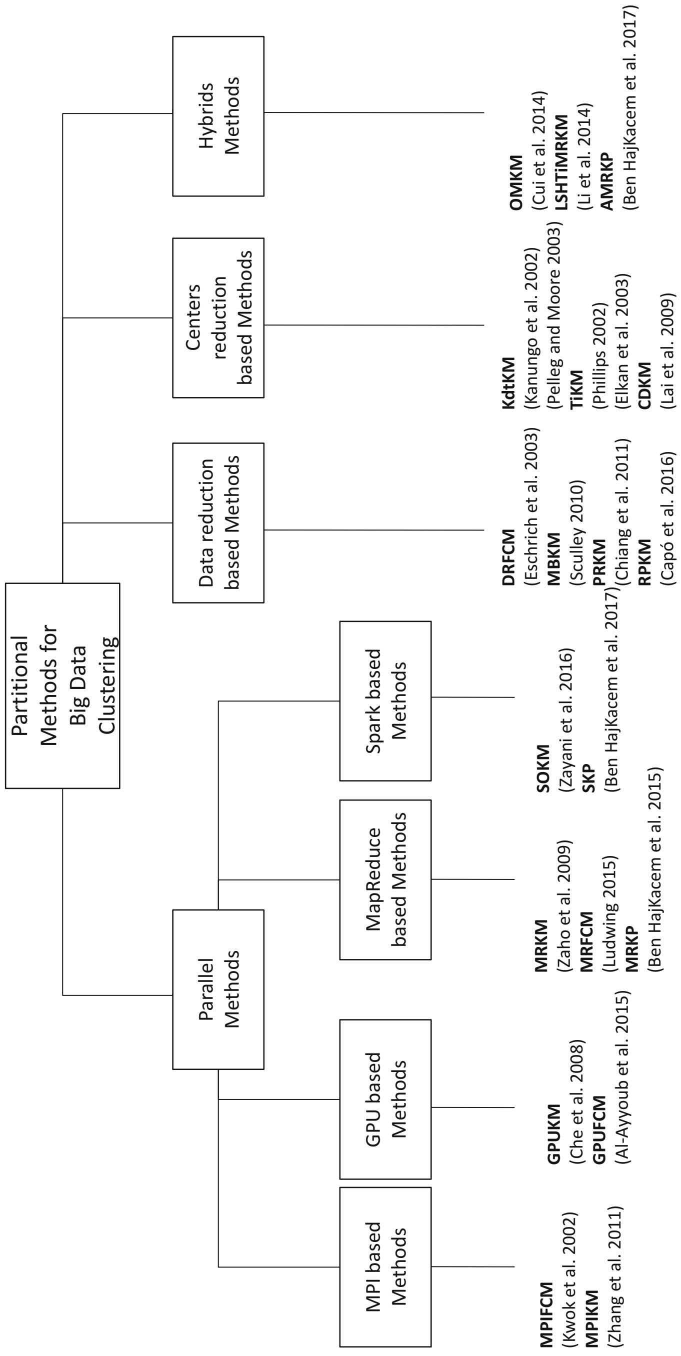

Partitional Methods for Big Data Clustering
Parallel Methods
Data reduction based Methods
Centers reduction based Methods
Hybrids Methods
MPI based Methods
GPU based Methods
MapReduce based Methods
Spark based Methods
MPIFCM (Kwok et al. 2002)
MPIKM (Zhang et al. 2011)
GPUKM (Che et al. 2008)
GPUFCM (Al-Ayyoub et al. 2015)
MRKM (Zaho et al. 2009)
MRFCM (Ludwing 2015)
MRKP (Ben HajKacem et al. 2015)
SOKM (Zayani et al. 2016)
SKP (Ben HajKacem et al. 2017)
DRFCM (Eschrich et al. 2003)
MBKM (Sculley 2010)
PRKM (Chiang et al. 2011)
RPKM (Capó et al. 2016)
KdtKM (Kanungo et al. 2002)
(Pelleg and Moore 2003)
TiKM (Phillips 2002)
(Elkan et al. 2003)
CDKM (Lai et al. 2009)
OMKM (Cui et al. 2014)
LSHTiMRKM (Li et al. 2014)
AMRKP (Ben HajKacem et al. 2017)

Fig. 1.1 Classification of scalable partitional clustering methods based on the used approach to improve scalability

the function *Broadcast* is used to send the same data or messages to all machines while the function *Barrier* is used to put a barrier and allows the synchronization between machines when running parallel programs. The main advantage of MPI is its master/slave architecture, where the slave machine can become the master of other processes. This can be extremely useful for dynamic resource allocation where the slaves have to process a large amount of data. Another advantage of MPI includes the data preserving, i.e., there is no need to read the same data many times, which can be preserved locally. Hence, MPI is well suited for iterative algorithms [44].

Several methods were proposed to implement clustering process using MPI framework. For example, Zhang et al. [52] proposed an MPI-based k-means (MPIKM). The main steps of this method are as follows: first, the input data is initially distributed among slaves. Then, the master selects k data points as initial centers, and broadcasts them to slaves. Then, each slave assigns corresponding data points to the nearest cluster by computing distances. After that, the master collects all information needed from slaves to update new cluster centers, and broadcasts them to slaves. Finally, this method iterates calling above steps several times until convergence. The MPIKM can reduce the time complexity of k-means from $O(n.k.l)$ to $O(s.k.l)$ where n the number of data points, k the number of clusters, l the number of iterations, and $s \ll n$ the maximal number of data points assigned to each slave node.

On the other hand, Kwok et al. [30] designed a parallelization of fuzzy c-means using MPI framework (MPIFCM). The master first divides the input data into splits and transfers them to slaves. Then, each slave receives the associated split, computes distances, and updates the membership matrix. After that, the master gets all information needed from the slaves to compute the new cluster centers, and sends them to slaves for the next iteration. Finally, the above steps are repeated until convergence. The MPIFCM decreases the time complexity of fuzzy c-means from $O(n.k^2.l)$ to $O(s.k^2.l)$.

Despite the efficiency of MPI framework to process large-scale data, it suffers from limit of the fault intolerance. MPI has no mechanism to handle faults. The failure of one machine in the network can cause the shutdown of the whole network. Hence, practitioners of MPI-based methods have to implement a fault tolerance mechanism within the master/slave architecture to manage faults or failures of machines in the network. The implementation of this mechanism is not a trivial task which explains the restricted use of this framework in real-world applications.

1.3.1.2 GPU-Based Methods

Graphics Processing Unit (GPU) is a specialized hardware designed to accelerate graphical operations such as video and image editing. Compute Unified Device Architecture (CUDA) is a parallel programming framework used to simplify the creation of parallel applications within GPU without delving into the hardware details. GPU has a large number of processing cores as compared to a Central

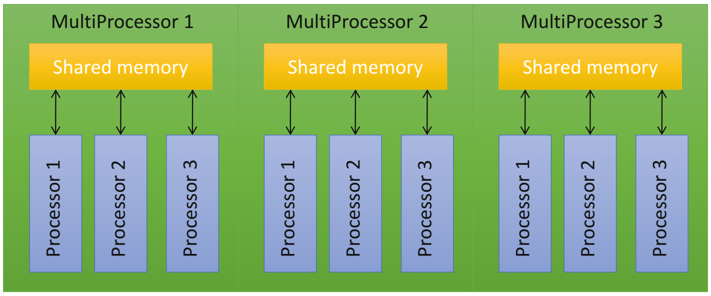


Fig. 1.2 GPU architecture with three multiprocessors and three streaming processors

Processing Unit (CPU). In addition, it provides two levels of parallelization. At the first level, GPU has several multiprocessors (MPs), and at the second level, each multiprocessor has several streaming processors (SPs). Following this configuration, GPU program is broken down into threads which execute on SPs, and these threads are aggregated together to form thread blocks which run on a multiprocessor. Figure 1.2 shows a GPU architecture with three multiprocessors and three streaming processors per block. Each thread within a block can communicate with each other and synchronize with other threads in the same block. Each of these threads has access to fast shared memory.

Several works were proposed to accelerate data clustering using GPU. For example, Che et al. [12] proposed a GPU-based k-means method (GPUKM). The main steps of this method are as follows, first, it uploads initial cluster centers to the shared memory of the GPU and the input dataset is partitioned and uploaded into each multiprocessor. Then, each multiprocessor calculates the distance from each corresponding data point and assigns it to the nearest cluster. After that, it calculates a local cluster center based on a subset of data points. Once all data points are assigned to cluster centers, CPU updates new cluster centers and again will upload them to multiprocessors. Finally, this method iterates calling the above steps several times until convergence. The GPUKM decreases the time complexity of k-means [19] from $O(n.k.l)$ to $O(g.k.l)$ where $g \ll n$ the maximal number of data points assigned to each multiprocessor.

Al-Ayyoub et al. [1] proposed the GPU fuzzy c-means method (GPUFCM). This method first stores initial positions of clusters to the shared memory. Then, it creates initial membership matrix and initial cluster centers from the input data. Then, each multiprocessor computes partial memberships by computing distances. Next, it computes the membership values via summation of partial memberships. After that, this method transfers summed membership values from GPU to CPU in order to compute new cluster centers. Finally, it moves to the next iteration. Similarly to GPUKM, the GPUFCM can reduce the time complexity of fuzzy c-means from $O(n.k^2.l)$ to $O(g.k^2.l)$.

Despite the attested performance of GPU for handling large-scale data, it suffers from the drawback of memory limits. For example, with a maximum of 12GB memory per GPU, it is not able to deal with terabyte data. When the data size exceeds the size of GPU memory, the performance of the GPU-based method decreases significantly. Hence, users have to configure the memory before implementing applications through GPU. Another drawback is the restricted number of software and algorithms that are available for GPUs.

1.3.1.3 MapReduce-Based Methods

MapReduce is a parallel programming framework used to process large-scale data across cluster of machines. It is characterized by its high transparency for programmers to parallelize algorithms in an easy and comfortable way. MapReduce is based on two phases namely *map* and *reduce*. Each phase has $< key/value >$ pairs as input and output. The map phase executes map functions to process in parallel each $< key/value >$ to generate a set of intermediate $< key'/value' >$ pairs. Then, this framework groups all intermediate values associated with the same intermediate key as a list (known as shuffle phase). The reduce phase executes reduce function to merge all intermediate values associated with the same intermediate key. Figure 1.3 illustrates the data flow of MapReduce framework. The inputs and outputs of MapReduce are stored in an associated distributed file system that is accessible from any machine of the used cluster. The implementation of the MapReduce framework is available in Hadoop [48]. Hadoop provides a distributed file system namely Hadoop Distributed File System (HDFS) which stores data on the machines. MapReduce has three major features: simple programming framework, linear scalability, and fault tolerance. These features make MapReduce a useful framework for large-scale data processing.

Several methods were proposed in the literature to fit clustering process through MapReduce. For instance, Zaho et al. [53] proposed a MapReduce-based k-means method (MRKM). Given an input dataset stored in HDFS, this method first divides the input dataset into splits where each split is associated with map function. Then, the map function assigns each data point of the associated split to the nearest cluster by computing distances. The reduce function then updates new cluster centers by calculating the average of data points present in each cluster. These new cluster centers are then written to the HDFS, to be used by the map function for the next iteration. Finally, the entire process is repeated until convergence. The time complexity of MRKM is evaluated by $O(m.k.l)$ where $m \ll n$ the maximal number of data points associated with the map function.

On the other side, Ludwing [35] proposed the parallelization of fuzzy c-means clustering using MapReduce framework (MRFCM). This method is based on two MapReduce jobs. The first MapReduce job calculates cluster center's matrix and the second MapReduce job calculates the distances, to be used to update the membership matrix. The map function of the first MapReduce job receives a chunk of data and a portion of the membership matrix and produces cluster center's sub-matrices. Then, the reduce function of the first MapReduce job merges sub-matrices into cluster center's matrix. The second MapReduce job compared to the first one involves more computations to be executed. During the map function, a chunk of

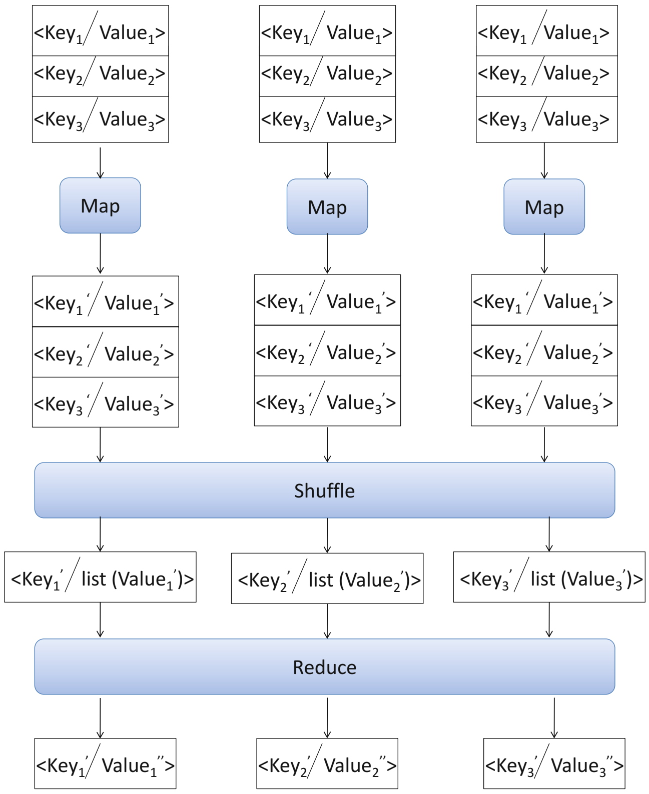


Fig. 1.3 Data flow of MapReduce framework

data is received and distance sub-matrices and membership matrices are computed. Then, the reduce function merges partition sub-matrices. The MRFCM decreases the time complexity of fuzzy c-means from $O(n.k^2.l)$ to $O(m.k^2.l)$.

Ben HajKacem et al. [4] proposed a MapReduce-based k-prototypes (MRKP) for clustering mixed large-scale data. First, MRKP creates a global variable that contains initial centers of the cluster. Then, the map function assigns each data point of the associated split to nearest cluster. Following each assignment, it computes an intermediate center for each cluster in order to optimize the calculation of new centers in reduce phase. This information consists of the sum of the numeric values and the frequencies of categorical values related to each cluster, which are then emitted to the reduce function. After that, the intermediate centers produced from map functions are merged by the reduce function, in order to update new cluster centers. Finally, new centers' values are stored in HDFS and the MRKP moves to the next iteration until convergence. Similarly to MRKM, the time complexity of MRKP is evaluated by $O(m.k.l)$.

Although MapReduce appears to be perfect for clustering large-scale data, it suffers from the inefficiency to run iterative algorithms [18]. The whole dataset must be read and written to HDFS for each iteration of the method. Therefore, many of

Table 1.1 Some operators in Spark

Operators	Meaning
map(**func**)	Iterates over each line in the RDD by calling **func** and returns only one element
flatmap(**func**)	Similar to map but returns a list of elements
mapparition(**func**)	Similar to map but runs separately on each partition (block) of the RDD
filtre(**func**)	Returns elements of the source RDD when **func** returns true
reducebykey(**func**)	Aggregates values with the same key using **func**
distinct()	Returns a new RDD that contains distinct elements of the source RDD

input/output (I/O) disk operations occur during each iteration and this significantly degrades the performance of MapReduce-based method.

1.3.1.4 Spark-Based Methods

Spark is a parallel framework for large-scale data processing designed to solve the MapReduces limitations. It was introduced as part of the Hadoop and it is designed to run with Hadoop, specially by reading data from HDFS. Spark is based on Resilient Distributed Datasets (RDDs), a special type of data structure used to parallelize computations in a transparent way. These parallel structures persist, reuse, and cache the results in memory. Moreover, Spark provides set of in-memory operators, beyond the standard MapReduce, with the aim of processing data more rapidly in distributed environments. Spark is faster up to 100× than MapReduce. Table 1.1 shows some operators of Spark which are used to implement parallel methods.

Several methods were proposed for Big data clustering within Spark framework. For instance, Ben HajKacem et al. [5] proposed a Spark-based k-prototypes clustering method for mixed large-scale data (SKP). The authors exploit in this method the in-memory operations of Spark to alleviate the consumption time of MRKP method. First, they create an RDD object with the input dataset formed by c chunks. The map function (mappartition) picks a chunk of dataset, executes the k-prototypes algorithm on that chunk, and emits the intermediate cluster centers as output. Then, the reduce function (reducebykey) takes set of intermediate centers, executes the k-prototypes algorithm again on them, and returns the final centers as output. For each map phase, there are m data points that must be processed using k-prototypes. Hence, map phase takes $O(m.k.l)$ time. In the reduce phase, the k-prototypes must be executed on set of intermediate centers which has $k.c$ items. Hence, the reduce phase needs $O(c.k^2.l)$ time. Given $c \ll m$, the overall time complexity of SKP is evaluated by $O(m.k.l)$.

Zayani et al. [51] proposed a parallelization of overlapping k-means method using Spark framework (SOKM). This method can perform parallel clustering processes leading to non-disjoint partitioning of data. The main steps of this method are as follows: first, they create an RDD object with input data formed by chunks.

Then, the map function (map) performs a local data assignment for each chunk in parallel in order to build the local membership matrix for set of data points within this chunk. Once local membership matrix is computed for each chunk, the second map function (map) takes these intermediate results and updates the local cluster centers. After that, the reduce function (reducebykey) computes global cluster centers based on intermediate cluster centers computed on each chunk. Finally, based on the evaluation of the global objective criterion, the proposed method reiterates all above-described steps while convergence is not reached. The time complexity of SOKM is evaluated by $O(m.k^2.l)$.

1.3.2 Data Reduction-Based Methods

This category of methods tries to reduce the number of data points when building clusters in order to accelerate the clustering process. Sculley [43] introduced a MinBatch k-means method (MBKM). Its main idea is to use small random batches of data points of a fixed size which can be stored in memory. The motivation behind this method is that random batches tend to have lower stochastic noise than individual data points. In each iteration, a new random sample from data is generated and used to build cluster centers. The cluster centers are then updated using a convex combination of values of cluster centers and data points by applying a dynamic rate. This dynamic rate is defined for each cluster and is evaluated by the inverse of the number of assigned data points. The MBKM decreases the time complexity of k-means from $O(n.k.l)$ to $O(b.k.l)$ where $b \ll n$ the batch size.

On the other hand, Capo et al. [9] proposed an alternative to the k-means for processing large-scale data called Recursive Partition k-means (RPKM). This method considers a sequence of partitions of the dataset, where the partition at iteration i is thinner than the partition at iteration $i - 1$. The idea behind this method is to approximate the k-means for the entire dataset by recursively applying a weighted version of k-means over a small number of subsets of data. The main steps of RPKM can be described as follows: first, the data is partitioned into a number of subsets where each subset is characterized by a representative and its corresponding weight. Second, a weighted version of k-means is applied over the set of representatives. From one iteration to the next, a more refined partition is built and the process is repeated using the obtained optimal cluster centers as initialization. The above steps are repeated until a stopping criterion is detected. The RPKM can reduce the time complexity of k-means from $O(n.k.l)$ to $O(p.k.l)$ where $p \ll n$ the partition size.

PRKM method proposed a Pattern Reduction algorithm for reducing the computation time of k-means [13]. Initially, the PRKM works exactly as k-means. Then, it continues to check whether it is time to start the pattern reduction algorithm. If the time to start is reached, the pattern reduction algorithm is applied. The pattern reduction algorithm is based on compressing and removing at each iteration the data points that are unlikely to change their membership thereafter. It can be divided into two procedures: the first is Pattern Compression and Removal (PCR), and the second is Pattern Assignment and Mean Update (PAMU). PCR requires a removal bound parameter which denotes the percentage of data points that are allowed to be compressed and removed. If the removal bound has not been reached, PCR first checks which data points in each cluster are near the mean and thus can be removed. We note that means is defined as the average of distances between data points and cluster centers. Then, PCR compresses and removes these data points by selecting one of data points to be removed as the representative data point and setting its value to the average of all data points removed. After that, PAMU reassigns each data point to the cluster to which it belongs first and then computes the new cluster centers. The PRKM can reduce the time complexity of k-means from $O(n.k.l)$ to $O(n.k)$.

Another method introduced a modified version of fuzzy c-means algorithm that uses Data Reduction algorithm to cluster large-scale data (DRFCM) [20]. The main idea of DRFCM is to reduce the data size before performing clustering. This method has two main phases, data reduction and data clustering. The data reduction phase consists of two steps which are precision reduction (an optional step) and aggregation. The aggregation step consists of combining identical data points together in the same weighted data points where the weights correspond to the number of aggregated data points. The clustering phase is devoted to apply the fuzzy c-means algorithm on the new weighted data points. To improve the running time of the aggregation step, the authors introduced a hashing for the aggregation step. Moreover, they optimize the calculation of cluster centers and membership matrix. The DRFCM decreases the time complexity of fuzzy c-means from $O(n.k^2.l)$ to $O(w.k^2.l)$ where $w \ll n$ the number of weighted data points generated by data reduction algorithm.

1.3.3 Centers Reduction-Based Methods

This category of methods aims to reduce the number of comparisons when looking for the nearest cluster centers which is the most time-consuming phase. Pelleg and Moore [39] and Kanungo et al. [28] proposed to accelerate k-means using kd-tree structure (KdtKM). A kd-tree is defined as a binary tree that partitions the space of the data and it is built by separating hyperplanes (typically axis-aligned) [42]. We note that k in kd-trees and in k-means are different. In k-means, it denotes the number of clusters while in kd-trees, denotes the dimension of the input data.

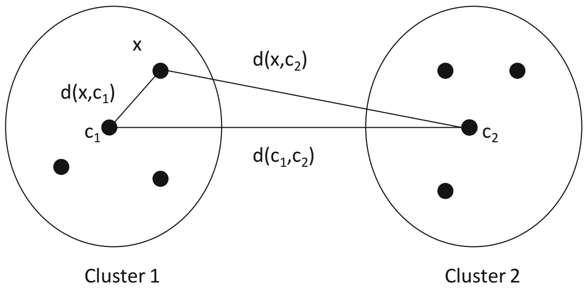
Pelleg and Moore [39] proposed to represent data as a kd-tree and used a pruning strategy to reduce the redundant comparisons between cluster centers and data points. This method initially creates the kd-tree from the input dataset. Then, it performs at each iteration a search through the tree for searching regions of the tree which are *owned* by a single center. The search begins with all k cluster centers at the root of the kd-tree. Then, it recursively goes through the depth of the tree while checking at each node that only one center *dominates* all the other centers. If so, it eliminates the dominated centers. In the case that only exists one dominated center, then the search stops and all data points below that node in the kd-tree are assigned to that center. However, when multiple centers remain, it recursively continues its search on child nodes. If it reaches a leaf, it performs distance computations between the remaining cluster centers and the data points at the leaf node. This method finds nearest cluster to each data point in $O(\log k)$ time. However, it requires $O(k.\log k)$ time in each iteration for building the kd-tree. Hence, it decreases the time complexity of k-means from $O(n.k.l)$ to $O(n.\log k.l + k.\log k)$. Although KdtKM method performs very well when dealing with large-scale data, it is not suitable for high-dimension data because of its exponential complexity regarding the number of dimensions of data. In addition, it requires an extra memory on the order of the input dataset for storing the kd-tree structure.

Other techniques are used to accelerate partitioning methods such as Triangle inequality-based k-means (TiKM) methods [17, 19, 23, 41]. For example, Phillips [41] used the triangle inequality to reduce the number of comparisons between cluster centers and data points. This method is motivated by the fact that k-means requires computing all distances between each of the cluster centers and data points. Many of these distance computations are redundant, because data points usually stay in the same cluster after first few iterations. This method uses the following triangle inequality to prove that if cluster center c_1 is close to data point x, and some other cluster center c_2 is far away from another cluster center c_1, then c_1 must be closer than c_2 to x.

Theorem 1.1 *Let x a data point and let c_1 and c_2 cluster centers (described in Fig. 1.4). If we know that $d(c_1,c_2) \geq 2 * d(x,c_1)$, then $d(x,c_1) \leq d(x,c_2)$ without having to calculate $d(x,c_2)$.*

In each iteration, the TiKM method uses the triangle inequality when looking for nearest cluster centers. Also, it requires computing distances between cluster centers. Hence, this method decreases the time complexity of k-means from $O(n.k.l)$ to $O((n.\gamma + k^2).l)$ where γ the average number of comparisons between data points and clusters selected at each iteration.

Fig. 1.4 The application of triangle inequality technique between data point x and the cluster centers c_1 and c_2



On the other hand, a Center Displacement k-means method (CDKM) is proposed to improve the efficiency of k-means [31]. This method first classifies cluster centers into static and active groups. Then, it uses the information of center displacements to reject impossible candidates when locking to the nearest cluster center. The CDKM decreases the time complexity of k-means from $O(n.k.l)$ to $O(n.k)$. We note that the time complexity of CDKM grows linearly with the data dimension in contrast to KdtKM methods which have the exponential dependence on the value of dimension. Moreover, CDKM obtains the same set of cluster centers as that produced by the conventional k-means.

1.3.4 Hybrids Methods

To deal with large-scale data, hybrids methods combine several acceleration techniques. The acceleration techniques are combined to win maximal efficiency when designing a clustering process for analyzing a large amount of data. Example of hybrids methods is the OMRKM proposed by Cui et al. [14]. This method proposed an optimized implementation of MapReduce-based k-means method using sampling. OMRKM consists of three MapReduce jobs namely: *Data sampling*, *Data clustering*, and *Data assignment*. The first MapReduce job is devoted generating a subset from input data using probability sampling. The second MapReduce job is concerned with clustering of the subset in order to obtain cluster centers. Once the cluster centers are computed, the third MapReduce job is executed to generate the partition matrix of input data by assigning each data point to the nearest cluster center. The OMRKM decreases the time complexity of k-means from $O(n.k.l)$ to $O(m+r.k.l)$ where $r \ll n$ the sample size.

On the other hand, Li et al. [33] proposed LSHTiMRKM method which is based on MapReduce framework, locality sensitive hashing [27] (LSH), and triangle inequality to improve the efficiency of k-means. This method consists of two MapReduce jobs namely: *Data skeleton* and *Data clustering*. The first job, data skeleton uses the LSH to map similar data points into buckets. Each bucket is represented by a weighted data point. Hence, the LSH technique is used to reduce the number of data points when building cluster centers. The second job, data clustering proposes an efficient implementation of scalable k-means [2]. This implementation is based on a pruning strategy to accelerate the iteration process by reducing unnecessary comparisons between cluster centers and data points. The pruning strategy begins by looking for the nearest cluster centers using triangle inequality. Then, it uses the local property of LSH to reduce distance computations between data points and cluster centers. Only centers in the bucket are evaluated. The LSHTiMRKM decreases the time complexity of k-means from $O(n.k.l)$ to $O(m+t.\gamma.l)$ where $t \ll n$ the number of buckets and $\gamma \ll k$ the average number of comparisons between data points and clusters selected at each iteration.

Ben HajKacem et al. [6] proposed an accelerated MapReduce-based k-prototypes for clustering mixed large-scale data (AMRKP). This method is based on a pruning strategy (PS) to reduce redundant distance computations between cluster centers and data points using triangle inequality technique. AMRKP introduces a KPPS algorithm which is based on applying a pruning strategy to k-prototypes. Initially, the KPPS works exactly as k-prototypes. Then, it continues to check whether it is time to start the pruning strategy. If the time to start is reached, the pruning strategy is applied. The pruning strategy requires at each iteration computing distances between centers and sorting them. Then, it evaluates the triangle inequality between data point and the centers in increasing order of distance to the assigned center of the previous iteration. If the pruning strategy reaches a center that does not satisfy the triangle inequality property, it can skip all the remaining centers and continue on to the next data point. The pruning strategy requires setting a pruning bound parameter (α) to denote the $\alpha\%$ subset of cluster centers that are considered when evaluating triangle inequality property. After that, this method distributes the KPPS algorithm within MapReduce framework. For this purpose, it first splits the input dataset into p chunks. Then, the map function picks a chunk of data, executes the KPPS algorithm on that chunk, and emits the intermediate cluster centers as the output. Then, the reduce function takes the set of intermediate centers, executes again the KPPS algorithm on them, and returns the final centers as output. The time complexity of KPPS algorithm is bounded between $O((n.\alpha.k{+}k^3).l)$ and $O((n.k{+}k^3).l)$ where α the pruning bound. The KPPS algorithm is applied twice: in the map phase and the reduce phases. In the map phase, each chunk involves running the KPPS algorithm on that chunk. This phase is evaluated by $O(m.\alpha.k{+}k^3).l)$ time. In the reduce phase, the KPPS algorithm must be executed on the set of intermediate centers which has $k.p$ data points. Hence, the reduce phase needs $O((m.\alpha.k{+}k^3).l)$ time. Given that $\frac{k.n}{p} \ll m$, the overall time complexity of the AMRKP is evaluated by $O((m.\alpha.k{+}k^3).l{+}\left(\frac{k.n}{p}.\alpha.k + k^3\right).l) \cong O((m.\alpha.k{+}k^3).l)$.

1.3.5 Summary of Scalable Partitional Clustering Methods for Big Data Clustering

This section gives an overview of the main characteristics of scalable partitional clustering methods presented in a comparative way. Table 1.2 summarizes these main characteristics. Our study is based on the following features of the methods: (1) type of data supported by each method, (2) the final results after acceleration regarding the conventional method, exact or approximate (3) time complexity, (4) space complexity, and (5) type of the used acceleration technique.

Parallel partitional clustering methods are divided into four categories MPI-based, GPU-based, MapReduce-based, and Spark-based methods. Before fitting clustering through parallel framework, it is important to consider some points. GPU suffers from the memory limitation. When the data size exceeds the size of the GPU memory, the performance decreases significantly the GPU-based method. For example, with a maximum of 12GB memory per GPU, it is not suitable to deal with terabyte data. Then, MPI has no mechanism to handle faults. The failure of one machine in the network can cause the shutdown of the whole network. Hence,

Table 1.2 The main characteristics of scalable partitional clustering methods for Big data clustering

Category	Method	References	Type of data	Results after acceleration	Time complexity	Space complexity	Acceleration technique
Parallel	MPIKM	Zhang et al. [52]	Numeric	Exact	$O(s.k.l)$	$O(s)$	MPI framework
	MPIFCM	Kwok et al. [30]	Numeric	Exact	$O(s.k^2.l)$.	$O(s)$	MPI framework
	GPUKM	Che et al. [12]	Numeric	Exact	$O(g.k.l)$	$O(g)$	GPU framework
	GPUFCM	Al-Ayoub et al. [1]	Numeric	Exact	$O(g.k.l)$	$O(g)$	GPU framework
	MRKM	Zaho et al. [53]	Numeric	Exact	$O(m.k.l)$	$O(m)$	MapReduce framework
	MRFCM	Ludwing [35]	Numeric	Exact	$O(m.k^2.l)$	$O(m)$	MapReduce framework
	MRKP	Ben HajKacem et al. [4]	Mixed	Exact	$O(m.k.l)$	$O(m)$	MapReduce framework
	SKP	Ben HajKacem et al. [5]	Mixed	Approximate	$O(m.k.l)$	$O(m)$	Spark framework
	SOKM	Zayani et al. [51]	Numeric	Exact	$O(m.k.l)$	$O(m)$	Spark framework
Data reduction based	MBKM	Scully [43]	Numeric	Approximate	$O(b.k.l)$	$O(n+b)$	Min batch
	RPKM	Capo et al. [9]	Numeric	Approximate	$O(p.k.l)$	$O(n+p)$	Recursive partition
	PRKM	Chiang et al. [13]	Numeric	Approximate	$O(n.k)$	$O(n)$	Pattern reduction
	DRFCM	Eschirh et al. [20]	Numeric	Exact	$O(w.k.l)$	$O(w)$	Data reduction
Centers reduction based	KdtKM	Pelleg and More [40]	Hard	Numeric	$O(n.\log k.l+k.\log k)$	$O(n+k.\log k)$	Kd-tree
	KdtKM	Kanungo et al. [28]	Numeric	Exact	$O((n.\gamma+k^2).l)$	$O(n+k^2)$	Kd-tree
	TiKM	Philips [41]	Numeric	Exact	$O((n.\gamma+k^2).l)$	$O(n+k^2)$	Triangle inequality
	TiKM	Elkan [19]	Numeric	Exact	$O((n.\gamma+k^2).l)$	$O(n+k^2)$	Triangle inequality
	CDKM	Lai et al. [31]	Numeric	Exact	$O(n.k)$	$O(n+k^2)$	Displacement center
Hybrids	OMRKM	Cui et al. [14]	Numeric	Approximate	$O(m.k+r.k.l)$	$O(n+r)$	MapReduce + Sampling
	LSHTiMRKM	Li et al. [33]	Numeric	Approximate	$O(m+t.\gamma.l)$	$O(n+t+k^2)$	MapReduce + LSH
	AMRKP	Ben HajKacem et al. [6]	Mixed	Approximate	$O(m.\alpha\%.k+k^3).l)$	$O(m+k^2)$	MapReduce + Triangle inequality

Notes: n: number of data points, k: number of clusters, l: number of iterations, s: number of data points assigned to salve, g: number of data points assigned to multiprocessor, m: number of data points assigned to map, b: batch size, p: partition size, w: number of weighted examples, γ: average number of comparisons, r: sample size, t: bucket size, $\alpha\%$: pruning bound practitioners have to implement some kind of fault tolerance mechanism within the program to overcome faults. The MapReduce framework looks better than MPI since it is characterized by simple programming framework, linear scalability, and fault tolerance. However, it is unsuitable to run iterative algorithms since at each iteration, the whole dataset must be read and written to disks and this results in high (I/O) operations. This significantly degrades the performance of MapReduce-based method. Finally, Spark framework is an alternative to MapReduce which is designed to overcome the disk I/O limitations and improve the performance of MapReduce framework. A recent survey has presented the different frameworks for Big data analytics and provides the advantages and drawbacks of each of these frameworks based on various metrics such as scalability, data I/O rate, fault tolerance, and iterative task support [37, 45].

Although all the described Big data partitional clustering methods offer for users an efficient analysis for large-scale data, some parameters need to be estimated before performing the learning. All the described methods require to configure the number of clusters in prior which is not a trivial task in real-life applications where the number of expected clusters is usually unknown. As a solution, one could use different model heuristics for determining the optimal number [34, 40]. For example, the user can test different clustering with an increased number of clusters and then, take the clustering having the best balance between the minimization of the objective function and the number of clusters. Furthermore, all the described Big data partitional methods need to initialize the clusters' centers. However, high-quality initialized centers are important for both accuracy and efficiency of conventional clustering methods. To overcome this problem, users can adopt random sampling method to obtain cluster centers or using initialization techniques which exploit the fact that a good clustering is relatively spread out [8, 10, 32]. Using center initialization techniques, the result of the presented methods always converges to a local optimum of the objective criterion, rather than the global optimum. To deal with this issue, users can combine conventional methods with heuristic techniques to prevent clustering results from falling into local optimum [3, 21, 29].

1.4 Empirical Evaluation of Partitional Clustering Methods for Large-Scale Data

We evaluate in this section the performance of the scalable partitional clustering methods. We select from each category of scalable Big data partitional clustering methods at least one representative method, such as MRKM [53] (parallel), SKP [5] (parallel), MBKM [43] (data reduction based), TiKM [41] (centers reduction based), and AMRKP [6] (hybrids). We have implemented all these methods with the Java version 8 programming language. For MBKM and TiKM, we used a single machine with 1-core 3.4 GHz $i5$ CPU and 4GB of memory while for MRKM, SKP, and AMRKP, we used a cluster of four machines where each machine has 1-core

2.3 GHz CPU and 1GB of memory. Concerning the implementation of MapReduce framework, we used Apache Hadoop version 1.6.0 for MRKM and AMRKP and we used Apache Spark version 1.6.2 for SKP.

The experiments are performed on simulated and real datasets. For simulated datasets, we generate two series of datasets with Gaussian distribution. The mean of generated data points of the Gaussian distribution is 350 and the sigma is 100. The datasets range from 5 to 10 million data points. Each data point is described using ten attributes. In order to simplify the names of simulated datasets, we will use the notations Sim5M and Sim10M to denote a generated dataset containing 5 and 10 million data points, respectively.

Concerning real datasets, we use the KDD Cup dataset (KDD), which consists of normal and attack connections simulated in a military network environment. The KDD dataset contains about 5 million connections. Each connection is described using 33 numeric attributes. The clustering process for these dataset detects the type of attacks among all connections. This dataset was obtained from UCI machine learning repository.[1] The second real dataset is the Household dataset (House), which contains the results of measurements of electric power consumption in household. The House dataset contains 1 million data points. Each data point is described using seven numeric attributes. The clustering process for these data identifies the types of electric consumption in household. This dataset was obtained from UCI machine learning repository.[2] Statistics of simulated and real datasets are summarized in Table 1.3.

To simplify the discussion of the empirical results in Table 1.4, we will use the following conventions, let ψ denotes one of the scalable partitional clustering methods, let T denote the running time, and let S denote the quality of the clustering results. The enhancement of the running time of scalable partitional clustering method (T_ψ) with respect to the running time of k-means (T_{KM}) in percentage is defined by:

$$\Delta_T = \frac{T_\psi - T_{\mathrm{KM}}}{T_{\mathrm{KM}}} * 100\% \tag{1.1}$$

For example, the enhancement of the running time of MRKM (T_{MRKM}) with respect to the running time of k-means is defined by:

Table 1.3 Summary of datasets

Dataset	Data points	Attributes	Domain
Sim5M	5,000,000	10	Simulated
Sim10M	10,000,000	10	Simulated
KDD	4,898,431	33	Intrusion detection
House	1,000,000	7	Electricity

[1] https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1998+Data.

[2] https://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption.

$$\Delta_T = \frac{T_{\text{MRKM}} - T_{\text{KM}}}{T_{\text{KM}}} * 100\% \tag{1.2}$$

It is important to note that as defined in Eq. 1.2, a more negative value of Δ_T implies a greater enhancement. The enhancement of the quality of scalable partitional clustering (S_ψ) with respect to the quality of k-means (S_{KM}) in percentage is defined by:

$$\Delta_S = \frac{S_\psi - S_{\text{KM}}}{S_{\text{KM}}} * 100\% \tag{1.3}$$

To evaluate the clustering quality, we use the Sum Squared Error (SSE) [49], which aims to measure the squared errors between each data point and the cluster center to which the data point belongs. SSE can be defined by:

$$\text{SSE} = \sum_{i=1}^{n} \sum_{j=1}^{k} d(c_j, x_i), \tag{1.4}$$

where x_i is the data point and c_j the cluster center.

Table 1.4 reports the obtained results of scalable partitional clustering methods compared to k-means on simulated and real datasets. We note that we test for each dataset three different number of clusters 10, 50, and 100 and we fix the number of iterations to 20. For other parameters, we consider the following: the batch size b to 10,000 for MBKM and the pruning bound α to 10 for AMRKP.

The analysis of the empirical results firstly shows that hybrids methods are significantly faster than all other methods since they use simultaneously several acceleration techniques to improve the efficiency of conventional k-means method. For instance, we can observe that TiKM reduces the running time by 35.33% while AMRKP reduces the running time by 85.60%. Hence, we can conclude that the combination of acceleration techniques is a good solution when dealing with large-scale data. The second conclusion concerns the parallel partitional clustering methods. The results show that SKP method is faster than MRKM method. For example, MRKM can reduce the running time of k-means by 75.15%. However, SKP can reduce the running time of k-means by 96.14%. This fact shows the benefits of Spark to execute clustering process in memory and reduce the I/O operations from disks. Hence, we can conclude that Spark framework is more suitable to cluster large-scale data than MapReduce framework. Third, empirical results show that the number of clusters has an impact on the performance of clustering especially the case of triangle inequality-based methods like TiKM and AMRKP. For example, TiKM reduces the running time of k-means by 53.66% when $k = 10$ while by 59.92% when $k = 100$. From this observation, we can mention that TiKM method performs well when dealing with large number of clusters. The fourth conclusion concerns the quality of the obtained results. The results show that MRKM and TiKM methods produce the same SSE values compared

Table 1.4 Empirical results on simulated and real datasets

		MRKM-KM		SKP-KM		MBKM-KM		TiKM-KM		AMRKP-KM	
Dataset	k	Δ_T	Δ_S	Δ_T	Δ_S	Δ_T	Δ_S	Δ_T	Δ_S	Δ_T	Δ_S
Sim5M	10	−75.15	0.00	−96.49	21.03	−98.38	−0.03	−35.33	0.00	−82.60	58.19
	50	−77.14	0.00	−96.25	28.36	−98.45	0.17	−29.33	0.00	−84.83	63.55
	100	−76.18	0.00	−96.77	154.84	−98.63	4.28	−28.64	0.00	−84.69	−65.70
Sim10M	10	−71.25	0.00	−96.52	20.71	−92.12	−0.03	−39.24	0.00	−84.45	58.40
	50	−71.25	0.00	−96.82	28.06	−98.11	1.27	−29.08	0.00	−83.76	63.56
	100	−71.25	0.00	−97.04	27.32	−98.87	−2.25	−36.17	0.00	−83.21	65.68
KDD	10	−75.12	0.00	−96.62	180.70	−96.83	45.27	−53.66	0.00	−85.82	98.50
	50	−75.12	0.00	−96.19	134.30	−98.12	73.04	−56.66	0.00	−87.12	98.14
	100	−75.12	0.00	−96.40	142.61	−98.20	73.88	−59.92	0.00	−88.55	98.50
House	10	−75.56	0.00	−92.99	25.87	−93.70	4.33	−59.32	0.00	−85.50	6.20
	50	−75.56	0.00	−93.48	25.74	−94.05	−2.47	−61.35	0.00	−83.96	7.58
	100	−75.56	0.00	−94.74	12.36	−94.80	0.88	−62.55	0.00	−83.14	6.74

to k-means method. Therefore, we can deduce that MapReduce framework and triangle inequality reduce the running time of conventional k-means method without affecting clustering results. However, MBKM method does not always converge to the same local solution because of the relatively small batch size, leading to significant variations in the SSE values.

As a summary, the empirical study allows us to draw the following conclusions:

- The hybrids methods are faster than all other methods since they use simultaneously several acceleration techniques to improve the efficiency of conventional methods. Therefore, the combination of acceleration techniques is a good solution when dealing with large-scale data.
- Spark framework is designed to overcome disk I/O limitations and supports iterative algorithms. Hence, it is more suitable than MapReduce framework for processing large-scale data.
- The triangle inequality-based methods perform well when the number of clusters is large. However, they require additional memory to store the distances between cluster centers.
- MapReduce and triangle inequality techniques can improve the efficiency of conventional clustering without affecting the final clustering results. While sampling-based methods like MBKM do not always converge to the same local solution since they use a subsample instead of the entire dataset.

1.5 Conclusion

We focused in this chapter the area of Big data clustering, for which we give a classification of the existing methods based on the used acceleration techniques. Our study is essentially based on partitional clustering methods. For that, we review the existing scalable Big data partitional methods in the literature, classified into four main categories: parallel methods, data reduction-based methods, centers reduction-based methods, and hybrids methods. We also provide theoretical and experimental comparisons of the existing Big data partitional methods.

At the end of this overview, we claim that Big data clustering has a growing interest in machine learning research since many real-world applications require a scalable partitioning from a large amount of data. Many current challenges in Big data clustering area motivate researchers to propose more effective and efficient learning process. For example, recent works are interested in the identification of groups from streaming large-scale data, or building clusters from uncertain large-scale data. Other works are interested in the identification of clustering from data having multiple representations. All these challenges within Big data clustering open exciting directions for future researchers.

References

1. M. Al-Ayyoub, A.M. Abu-Dalo, Y. Jararweh, M. Jarrah, M. Al Sa'd, A GPU-based implementations of the fuzzy C-means algorithms for medical image segmentation. J. Supercond. **71**(8), 3149–3162 (2015)
2. B. Bahmani, B. Moseley, A. Vattani, R. Kumar, S. Vassilvitskii, Scalable k-means++. Proc. VLDB Endow. **5**(7), 622–633 (2012)
3. S. Bandyopadhyay, U. Maulik, An evolutionary technique based on K-means algorithm for optimal clustering in RN. Inform. Sci. **146**(1), 221–237 (2002)
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*, pp. 1–7 (2015)
5. M.A. Ben HajKacem, C.E. Ben N'cir, N. Essoussi, KP-S: a spark-based design of the K-prototypes clustering for big data, in *Proceedings of ACS/IEEE International Conference on Computer Systems and Applications*, pp. 1–7 (2017)
6. M.A. Ben HajKacem, C.E. Ben N'cir, N. Essoussi, One-pass MapReduce-based clustering method for mixed large scale data. J. Intell. Inf. Syst. 1–18 (2017)
7. J.C. Bezdek, R. Ehrlich, W. Full, FCM: the fuzzy c-means clustering algorithm. Comput. Geosci. **10**(2–3), 191–203 (1984)
8. P.S. Bradley, U.M. Fayyad. Refining initial points for K-means clustering, in *Proceeding ICML '98 Proceedings of the Fifteenth International Conference on Machine Learning*, vol. 98, pp. 91–99 (1998)
9. M. Capó, A. Pérez, J.A. Lozano, An efficient approximation to the k-means clustering for massive data. Knowl.-Based Syst. **117**, 56–69 (2017)
10. 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)
11. O. Chapelle, B. Scholkopf, A. Zien, Semi-supervised learning (Chapelle, O. et al., Eds.; 2006)[Book reviews]. IEEE Trans. Neural Netw. **20**(3), 542–542 (2009)
12. S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, K. Skadron, A performance study of general-purpose applications on graphics processors using CUDA. J. Parallel Distrib. Comput. **68**(10), 1370–1380 (2008)
13. M.C. Chiang, C.W. Tsai, C.S. Yang, A time-efficient pattern reduction algorithm for k-means clustering. Inform. Sci. **181**(4), 716–731 (2011)
14. X. Cui, P. Zhu, X. Yang, K. Li, C. Ji, Optimized big data K-means clustering using MapReduce. J. Supercomput. **70**(3), 1249–1259 (2014)
15. J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
16. R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*. Wiley, Hoboken (2012)
17. J. Drake, G. Hamerly, Accelerated k-means with adaptive distance bounds, in *5th NIPS Workshop on Optimization for Machine Learning*, pp. 42–53 (2012)
18. J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.H. Bae, J. Qiu, G. Fox, Twister: a runtime for iterative MapReduce, in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 810–818 (ACM, 2010)
19. C. Elkan, Using the triangle inequality to accelerate k-means, in *Proceeding ICML'03 Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, vol. 1(3) (2003), pp. 147–153
20. S. Eschrich, J. Ke, L.O. Hall, D.B. Goldgof, Fast accurate fuzzy clustering through data reduction. IEEE Trans. Fuzzy Syst. **11**(2), 262–270 (2003)
21. 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)
22. A. Fahad, N. Alshatri, Z. Tari, A. Alamri, I. Khalil, A.Y. Zomaya, . . ., A. Bouras, A survey of clustering algorithms for big data: taxonomy and empirical analysis. IEEE Trans. Emerg. Top. Comput. **2**(3), 267–279 (2014)

23. G. Hamerly, C. Elkan, Alternatives to the k-means algorithm that find better clusterings, in *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, pp. 600–607 (ACM, New York, 2002)
24. J. Han, J. Pei, M. Kamber, *Data Mining: Concepts and Techniques* (Elsevier, New York, 2011)
25. Z. Huang, Clustering large data sets with mixed numeric and categorical values, in *Proceedings of the 1st Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 21–34 (1997)
26. Z. Huang, Extensions to the k-means algorithm for clustering large data sets with categorical values. Data Min. Knowl. Disc. **2**(3), 283–304 (1998)
27. P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pp. 604–613 (1998)
28. T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, A.Y. Wu, An efficient k-means clustering algorithm: analysis and implementation. IEEE Trans. Pattern Anal. Mach. Intell. **24**(7), 881–892 (2002)
29. K. Krishna, M.N. Murty, Genetic K-means algorithm. IEEE Trans. Syst. Man Cybern. B Cybern. **29**(3), 433–439 (1999)
30. T. Kwok, K. Smith, S. Lozano, D. Taniar, Parallel fuzzy c-means clustering for large data sets, in *Euro-Par 2002 Parallel Processing*, pp. 27–58 (2002)
31. J.Z. Lai, T.J. Huang, Y.C. Liaw, A fast k-means clustering algorithm using cluster center displacement. Pattern Recogn. **42**(11), 2551–2556 (2009)
32. M. Laszlo, S. Mukherjee, A genetic algorithm using hyper-quadtrees for low-dimensional k-means clustering. IEEE Trans. Pattern Anal. Mach. Intell. **28**(4), 533–543 (2006)
33. Q. Li, P. Wang, W. Wang, H. Hu, Z. Li, J. Li, An efficient k-means clustering algorithm on MapReduce, in *Proceedings of Database Systems for Advanced Applications*, pp. 357–371 (2014)
34. A. Likas, N. Vlassis, J.J. Verbeek, The global k-means clustering algorithm. Pattern Recogn. **36**(2), 451–461 (2003)
35. S.A. Ludwig, MapReduce-based fuzzy c-means clustering algorithm: implementation and scalability. Int. J. Mach. Learn. Cybern. **6**, 1–12 (2015)
36. J. MacQueen, Some methods for classification and analysis of multivariate observations, in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* **14**(1), 281–297 (1967)
37. A. Mohebi, S. Aghabozorgi, T. Ying Wah, T. Herawan, R. Yahyapour, Iterative big data clustering algorithms: a review. Softw. Pract. Exp. **46**(1), 107–129 (2016)
38. J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, J.C. Phillips, GPU computing. Proc. IEEE **96**(5), 879–899 (2008)
39. D. Pelleg, A. Moore, Accelerating exact k-means algorithms with geometric reasoning, in *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 277–281. (ACM, New York, 1999)
40. D. Pelleg, A.W. Moore, X-means: extending k-means with efficient estimation of the number of clusters, in *Proceedings of the 17th International Conference on Machine Learning*, vol. 1, pp. 727–734 (2000)
41. S.J. Phillips, Acceleration of k-means and related clustering algorithms, in *Algorithm Engineering and Experiments*, pp. 166–177 (Springer, Berlin, 2002)
42. S.J. Redmond, C. Heneghan, A method for initialising the K-means clustering algorithm using kd-trees. Pattern Recogn. Lett. **28**(8), 965–973 (2007)
43. D. Sculley, Web-scale k-means clustering, in *Proceedings of the 19th International Conference on World Wide Web*, pp. 1177–1178 (ACM, New York, 2010)
44. O. Sievert, H. Casanova, A simple MPI process swapping architecture for iterative applications. Int. J. High Perform. Comput. Appl. **18**(3), 341–352 (2004)
45. D. Singh, C.K. Reddy, A survey on platforms for big data analytics. J. Big Data **2**(1), 8 (2015)
46. M. Snir, *MPI—The Complete Reference: The MPI Core*, vol. 1 (MIT Press, Cambridge, 1998), pp. 22–56

47. A. Vattani, K-means requires exponentially many iterations even in the plane. Discret. Comput. Geom. **45**(4), 596–616 (2011)
48. T. White, *Hadoop: The Definitive Guide* (O'Reilly Media, Sebastopol, 2012)
49. R. Xu, D.C. Wunsch, Clustering algorithms in biomedical research: a review. IEEE Rev. Biomed. Eng. **3**, 120–154 (2010)
50. M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: cluster computing with working sets. HotCloud **10**(10–10), 95 (2010)
51. A. Zayani, C.E. Ben N'Cir, N. Essoussi, Parallel clustering method for non-disjoint partitioning of large-scale data based on spark framework, in *Proceedings of IEEE International Conference on Big Data*, pp. 1064–1069 (IEEE, Piscataway, 2016)
52. J. Zhang, G. Wu, X. Hu, S. Li, S. Hao, A parallel k-means clustering algorithm with MPI, in *Proceedings of Fourth International Symposium on Parallel Architectures, Algorithms and Programming*, pp. 60–64 (2011)
53. W. Zhao, H. Ma, Q. He, Parallel k-means clustering based on MapReduce, in *Proceedings of Cloud Computing*, pp. 674–679 (2009)