J Intell Inf Syst (2019) 52:619–636
DOI 10.1007/s10844-017-0472-5

One-pass MapReduce-based clustering method for mixed large scale data

Mohamed Aymen Ben HajKacem[1] · Chiheb-Eddine Ben N'cir[1] · Nadia Essoussi[1]

Received: 20 December 2016 / Revised: 17 April 2017 / Accepted: 4 July 2017 /
Published online: 15 July 2017

Abstract Big data is often characterized by a huge volume and a mixed types of attributes namely, numeric and categorical. K-prototypes has been fitted into MapReduce framework and hence it has become a solution for clustering mixed large scale data. However, k-prototypes requires computing all distances between each of the cluster centers and the data points. Many of these distance computations are redundant, because data points usually stay in the same cluster after first few iterations. Also, k-prototypes is not suitable for running within MapReduce framework: the iterative nature of k-prototypes cannot be modeled through MapReduce since at each iteration of k-prototypes, the whole data set must be read and written to disks and this results a high input/output (I/O) operations. To deal with these issues, we propose a new one-pass accelerated MapReduce-based k-prototypes clustering method for mixed large scale data. The proposed method reads and writes data only once which reduces largely the I/O operations compared to existing MapReduce implementation of k-prototypes. Furthermore, the proposed method is based on a pruning strategy to accelerate the clustering process by reducing the redundant distance computations between cluster centers and data points. Experiments performed on simulated and real data sets show that the proposed method is scalable and improves the efficiency of the existing k-prototypes methods.

Keywords K-prototypes · One-pass MapReduce · Large scale data · Mixed data · Pruning strategy

✉ Mohamed Aymen Ben HajKacem
medaymen.hajkacem@gmail.com

Chiheb-Eddine Ben N'cir
chiheb.benncir@isg.rnu.tn

Nadia Essoussi
nadia.essoussi@isg.rnu.tn

[1] Institut Supérieur de Gestion de Tunis, LARODEC, Université de Tunis, 41 Avenue de la liberté, cité Bouchoucha, 2000 Le Bardo, Tunisia

1 Introduction

Large volume of data are being collected from different sources and there is a high demand for methods and tools that can efficiently analyse these large volume of data referred to as Big data analysis. Big data usually refers to three mains characteristics also called the three Vs (Gorodetsky 2014) which are respectively *Volume*, *Variety* and *Velocity*. Volume refers to the large scale data, Variety indicates the mixed types of data such as numeric, categorical and text data. Velocity refers to the speed at which data is generated and processed (Gandomi and Haider 2015). Several frameworks have been proposed for processing Big data. The most well-known is MapReduce framework (Dean and Ghemawat 2008). MapReduce is initially developed by Google and it is designed for processing Big data by exploiting the parallelism among a cluster of machines. MapReduce has three major features: simple programming framework, linear scalability and fault tolerance. These features make MapReduce an useful framework for Big data processing (Lee et al. 2012).

Clustering is an important technique in machine learning, which has been used to organize data into groups of similar data points called also clusters. Examples of clustering methods categories are hierarchical methods, density-based methods, grid-based methods, model-based methods and partitional methods (Jain et al. 1999). These clustering methods were well used in several applications such as intrusion detection (Tsai et al. 2009; Wang et al. 2010), customer segmentation (Liu and Ong 2008), document clustering (Ben N'Cir and Essoussi 2015; Hussain et al. 2014), image organization (Ayech and Ziou 2015; Du et al. 2015). In fact, conventional clustering methods are not suitable when dealing with large scale data. This is explained by the high computational cost of these methods which require unrealistic time to build the groupings. Furthermore, some clustering methods like hierarchical clustering cannot be applied to Big data because of its quadratic complexity time. Hence, clustering methods with linear time complexity should be used to handle large scale data.

On the other hand, Big data is often characterized by the mixed types of data including numeric and categorical. K-prototypes is one of the most well-known clustering methods to deal with mixed data, because of its linear time complexity (Ji et al. 2013). It has been successfully fitted into MapReduce framework in order to perform the clustering of mixed large scale data Ben Haj Kacem et al. (2015, 2016). However, the proposed methods have some considerable shortcomings. The first shortcoming is inherit from the conventional k-prototypes method, which requires computing all distances between each of the cluster centers and the data points. Many of these distance computations are redundant, because data points usually stay in the same cluster after first few iterations. The second shortcoming is the result of inherent conflict between MapReduce and k-prototypes. K-prototypes is an iterative algorithm which requires to perform some iterations for producing optimal results. In contrast, MapReduce has a significant problem with iterative algorithms (Ekanayake et al. 2010). As a consequence, the whole data set must be loaded from the file system into the main memory at each iteration. Then, after it is processed, the output must be written to the file system again. Therefore, many of I/O disk operations occur during each iteration and this decelerates the running time.

In order to overcome the mentioned shortcomings, we propose in this paper a new one-pass **A**ccelerated **M**ap**R**educe-based **K**-**P**rototypes clustering method for mixed large scale data, referred to as AMRKP. The proposed method is based on a pruning strategy to accelerate the clustering process by reducing the redundant distance computations between cluster centers and data points. Furthermore, the proposed method reads the data set only once in contrast to existing MapReduce implementation of k-prototypes. Our solution decreases the time complexity of k-prototypes (Huang 1997) from $O(n.k.l)$ to $O((n.\alpha\%.k + k^3).l)$ and the I/O complexity of MapReduce-based k-prototypes method (Ben Haj Kacem et al. 2015) from $O(\frac{n}{p}.l)$ to $O(\frac{n}{p})$ where n the number of data points, k the number of clusters, $\alpha\%$ the pruning heuristic, l the number of iterations and p the number of chunks.

The rest of this paper is organized as follows: Section 2 provides related works which propose to deal with large scale and mixed data. Then, Section 3 presents the k-prototypes method and MapReduce framework. After that, Section 4 describes the proposed AMRKP method while Section 5 presents experiments that we have performed to evaluate the efficiency of the proposed method. Finally, Section 6 presents conclusion and future works.

2 Related works

Given that data are often described by mixed types of attributes such as, numeric and categorical, a pre-processing step is usually required to transform data into a single type since most of proposed clustering methods deal with only numeric or categorical attributes. However, transformation strategies is often time consuming and produces information loss, leading to inaccurate clustering results (Ahmad and Dey 2007). Therefore, clustering methods have been proposed in the literature to perform the clustering of mixed data without pre-processing step (Ahmad and Dey 2007; Ji et al. 2013; Huang 1997; Li and Biswas 2002). For instance, Li and Biswas (2002) introduced the Similarity-Based Agglomerative Clustering called SBAC, which is an hierarchical agglomerative algorithm for mixed data. Huang (1997) proposed k-prototypes method which integrates k-means and k-modes methods to cluster numeric and categorical data. Ji et al. (2013) proposed an improved k-prototypes to deal with mixed type of data. This method introduced the concept of the distributed centroid for representing the prototype of categorical attributes in a cluster. Among the latter discussed methods, k-prototypes remains one of the most popular method to perform clustering from mixed data because of its efficiency (Ji et al. 2013). Nevertheless, it can not scale with huge volume of mixed data.

To deal with large scale data, several clustering methods which are based on parallel frameworks have been designed in the literature (Bahmani et al. 2012; Hadian and Shahrivari 2014; Kim et al. 2014; Ludwig 2015; Shahrivari and Jalili 2016; Zhao et al. 2009). Most of these methods use the MapReduce framework. For instance, Zhao et al. (2009) have implemented k-means method through MapReduce framework. Bahmani et al. have proposed a scalable k-means (Bahmani et al. 2012) that extends k-means++ technique for initial seeding. Shahrivari and Jalili (2016) have proposed a single-pass and linear time MapReduce-based k-means method. Kim et al. (2014) have proposed parallelizing density-based clustering with MapReduce. A parallel implementation of fuzzy c-means algorithm into MapReduce framework is presented in Ludwig (2015).

On the other hand, several methods have used the triangle inequality property to improve the efficiency of the clustering application (He et al. 2010; Nanni 2005). The triangle inequality is used as an exact mathematical property method in order to reduce the number of redundant unnecessary distance computations. In fact, the triangle inequality property is based on the hypothesis that is unnecessary to evaluate distances between a data point and clusters centers which are not closer to the old assigned center. For example, He et al. (2010), have proposed an accelerated Two-Threshold Sequential Algorithm Scheme (TTSAS). This method avoids unnecessary distance calculations by applying the triangle inequality. Nanni (2005) have exploited triangle inequality property of metric space to accelerated the hierarchical clustering method (single-link and complete-link). Although obtained clusters with these discussed methods are exactly the same as the standard ones, they require evaluating the triangle inequality property for all the set of clusters' center. Furthermore, these methods keep the entire data in the main memory for processing which reduces their performance on large data sets.

Although the later discussed methods offer for users an efficient analysis of large scale data, they can not support the mixed types of data and are limited to only numeric attributes. In order to perform the clustering of mixed large scale data, Ben Haj Kacem et al. (2015) proposed a parallelization of k-prototypes method through MapReduce framework. This method iterates two main steps until convergence: the step of assigning each data point to the nearest cluster center and the step of updating cluster centers. These two steps are implemented in map and reduce phase respectively. Although this method proposes an effective solution for clustering mixed large scale data, it has some considerable shortcomings. First, k-prototypes requires computing all distances between each of the cluster centers and the data points. However, many of these distance computations are redundant. Second, k-prototypes is not suitable for running within MapReduce framework since during each iteration of k-prototypes, the whole data set must be read and written to disks and this requires lots of I/O operations.

3 Preliminaries

This section first presents the k-prototypes method, then presents the MapReduce framework.

3.1 K-prototypes method

Given a data set $X = \{x_1 \dots x_n\}$ containing n data points, described by m_r numeric attributes and m_t categorical attributes, the aim of k-prototypes (Huang 1997) is to find k clusters by minimizing the following objective function:

$$J = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij} d(x_i, c_j), \tag{1}$$

where $u_{ij} \in \{0, 1\}$ is an element of the partition matrix U_{n*k} indicating the membership of data point i in cluster j, $c_j \in C = \{c_1 \dots c_k\}$ is the center of the cluster j and $d(x_i, c_j)$ is the dissimilarity measure which is defined as follows:

$$d(x_i, c_j) = \sum_{r=1}^{m_r} \sqrt{(x_{ir} - c_{jr})^2} + \sum_{t=1}^{m_t} \delta(x_{it}, c_{jt}), \tag{2}$$

where x_{ir} represents the value of numeric attribute r and x_{it} represents the value of categorical attribute t for data point i. c_{jr} represents the mean of numeric attribute r and cluster j, which can be calculated as follows:

$$c_{jr} = \frac{\sum_{i=1}^{|c_j|} x_{ir}}{|c_j|}, \tag{3}$$

where $|c_j|$ the number of data points assigned to cluster j. c_{jt} represents the most common value (mode) for categorical attributes t and cluster j, which can be calculated as follows:

$$c_{jt} = a_t^h, \tag{4}$$

where

$$f(a_t^h) \geq f(a_t^z), \quad \forall z, \quad 1 \leq z \leq m_c, \tag{5}$$

where $a_t^z \in \left\{a_t^1 \ldots a_t^{m_c}\right\}$ is the categorical value z and m_c is the number of categories of categorical attribute t. $f(a_t^z) = |\left\{x_{it} = a_t^z | p_{ij} = 1\right\}|$ is the frequency count of attribute value a_t^z. For categorical attributes, $\delta(p, q) = 0$ when $p = q$ and $\delta(p, q) = 1$ when $p \neq q$. It is easy to verify that the dissimilarity measure given in (2), is a *metric* distance since it satisfies the non-negativity, symmetry, identity and triangle inequality property as follows (Han et al. 2011; Ng et al. 2007):

1. d(x_i, x_j)>0 $\forall\, x_i, x_j \in$ X (Non-negativity)
2. d(x_i, x_j)=d(x_j, x_i) $\forall\, x_i, x_j \in$ X (Symmetry)
3. d(x_i, x_j)=0 $\Leftrightarrow x_i{=}x_j$ $\forall\, x_i, x_j \in$ X (Identity)
4. d(x_i, x_z)+d$(x_z, x_j)$$\geq$ d(x_i, x_j) $\forall\, x_i, x_j$ and $x_z \in$ X (Triangle inequality)

The main algorithm of k-prototypes method is described by Algorithm 1.

Algorithm 1 The main algorithm of the k-prototypes method

```
Input: $X$: Data set, $k$: number of clusters
Output: Cluster centers
begin
    1. Choose $k$ initial cluster centers randomly from $X$.
    2. Assign each data point in $X$ to the closest center by computing distances using
    Equation (2).
    3. Update the cluster centers using Equations (3) and (4).
    4. If the new cluster centers and the previous ones are the same, then terminate;
    otherwise, return to Step 2.
end
```

3.2 MapReduce framework

MapReduce (Dean and Ghemawat 2008) is a parallel programming framework designed to process large scale data across cluster of machines. It is characterized by its highly transparency for programmers, which allows to parallelize algorithms in a easy and comfortable way. The algorithm to be parallelized needs to be specified by only two phases namely map and reduce. Each phase has $< key/value >$ pairs as input and output. The map phase takes in parallel each $< key/value >$ pair and generates one or more 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 takes this list as input for generating final values. The whole process can be summarized as follows:

$$\text{Map}(< key/value >) \quad \rightarrow \quad \text{list}(< key'/value' >)$$
$$\text{Reduce}(< key'/\text{list}(value') >) \quad \rightarrow \quad \text{list}(< key'/value'' >)$$

Figure 1 illustrates the data flow of MapReduce framework. Note that the inputs and outputs of MapReduce are stored in an associated distributed file system that is accessible from any machine of the used cluster. As we mentioned earlier, MapReduce has a significant problems with iterative algorithms (Ekanayake et al. 2010). Hence, many of I/O operations occur during each iteration and this decelerates the running time. Several solutions have been proposed for extending the MapReduce framework to support iterations such as Twister (Ekanayake et al. 2010), Spark (Zaharia et al. 2010) and Phoenix (Talbot et al.

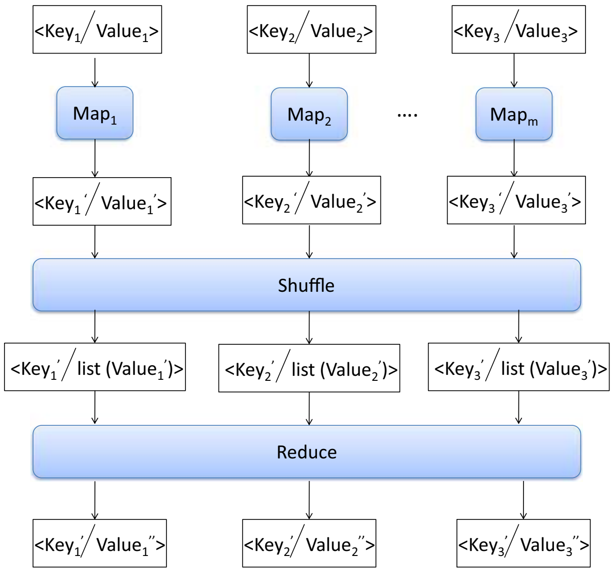

Fig. 1 Data flow of MapReduce framework

2011). These solutions excel when data can fit in memory because memory access latency is lower. However, Hadoop MapReduce (White 2012) can be an economical option because of Hadoop as a Service offering (HaaS), availability and maturity. In fact, the motivation of our work is to propose a new disk-efficient implementation of a clustering method for mixed large volume of data. As a solution, we propose the one pass disk implementation of k-prototypes within Hadoop MapReduce. This implementation reads and writes data only once, in order to reduce largely the I/O operations.

4 One-pass accelerated MapReduce-based k-prototypes clustering method for mixed large scale data

This section presents the proposed pruning strategy and the one-pass parallel implementation of our solution, followed by the parameters selections and the complexity analysis of the proposed method.

4.1 Pruning strategy

In order to reduce the number of unnecessary distance computations, we propose a pruning strategy, which requires a pruning heuristic $\alpha\%$ to denote the $\alpha\%$ subset of cluster centers that are considered when evaluating triangle inequality property. The proposed pruning strategy is based on the following assumptions:

- data points usually stay in the same cluster after first few iterations.
- If an assignment of an object has changed from one cluster to another, then the new cluster center is close to the old assigned center.

This strategy is inspired from marketing where there are a host of products in a particular category that they are aware of, but only a few they would actively consider for purchasing

(Eliaz and Spiegler 2011). This is known as consideration set. While all of the products that a consumer will think of when it's time to make a purchasing decision (known as the awareness set), the consideration set includes only those products they would consider a realistic option for purchase. In our case, we define the consideration set of centers as the set of $\alpha\%$ closest centers from a given old assigned center. The selected assignment of a data object will be the closest center among the "consideration set" of cluster centers.

The pruning strategy requires at each iteration computing distances between centers and sorting them. Then, it evaluates the triangle inequality property until the property is not satisfied or all centers in the subset of selected centers (the consideration set) have been evaluated. In other words, it evaluates the following theorem 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 continues on to the next data point.

Theorem 1 *Let x_i a data point, c_1 its cluster center of the previous iteration and c_2 another cluster center. If we know that $d(c_1, c_2) \geq 2* d(x_i, c_1) \Rightarrow d(x_i, c_1) \leq d(x_i, c_2)$ without having to calculate $d(x_i, c_2)$.*

Proof According to triangle inequality, we know that $\mathrm{d}(c_1, c_2) \leq \mathrm{d}(x_i, c_1)+\mathrm{d}(x_i, c_2) \Rightarrow \mathrm{d}(c_1, c_2)-\mathrm{d}(x_i, c_1) \leq \mathrm{d}(x_i, c_2)$. Consider the left-hand side $\mathrm{d}(c_1, c_2)-\mathrm{d}(x_i, c_1) \geq 2* \mathrm{d}(x_i, c_1)-\mathrm{d}(x_i, c_1) = \mathrm{d}(x_i, c_1) \Rightarrow \mathrm{d}(x_i, c_1) \leq \mathrm{d}(x_i, c_2)$. □

Notably, setting the pruning heuristic $\alpha\%$ too small may decrease the accuracy rate whereas setting the $\alpha\%$ to too large imposes a limit on the number of distance computations that can be reduced. The impact of the pruning heuristic on the performance of the proposed method will be discussed in Section 5.5. Algorithm 2 describes the different steps of the pruning strategy.

Algorithm 2 The pruning strategy

```
Input: $X$: Data set, $C$: Cluster centers
Output: $U$: Partition matrix
begin
    foreach $c_j \in C$ do
        Compute the distance between $c_j$ and other cluster centers.
        Sort the centers in ascending order of distance to $c_j$.
    foreach $x_i \in X$ do
        Let $c_p$ the cluster center of the previous iteration of $x_i$.
        Compute the distance between $x_i$ and $c_p$.
        Let $C_p^\alpha$ the first $\alpha$% cluster centers closet to $c_j$.
        foreach $c_j \in C_p^\alpha$ do
            if $d(c_j, c_p) < 2*d(x_i, c_p)$ then
                Compute the distance between $x_i$ and $c_j$.
            else
                break
        Let $c_o$ the closet cluster center of $x_i$.
        Assign the data point $x_i$ to $c_o$.
        $u_{io} \leftarrow 1$
end
```

Algorithm 3 gives the main algorithm of k-prototypes with pruning strategy which we call it KP+PS algorithm in the rest section of the paper. Initially, the KP+PS algorithm works exactly the same 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.

Algorithm 3 The main algorithm of the k-prototypes with pruning strategy (KP+PS algorithm)

```
Input: $X$: Data set, $k$: number of clusters
Output: Cluster centers
begin
    1. Choose $k$ initial cluster centers randomly from $X$.
    2. If the time to start is reached, apply the pruning strategy; otherwise assign each data
    point in $X$ to the nearest cluster center.
    3. Update the cluster centers using Equations (3) and (4).
    4. If the new cluster centers and the previous ones are the same, then terminate;
    otherwise, return to Step 2.
end
```

4.2 Parallel implementation

The proposed AMRKP method towards handling mixed large scale data consists of the parallelization of KP+PS algorithm based on the MapReduce framework. For this purpose, we first split the input data set into p chunks. Then, each chunk is processed independently in parallel way by its assigned machine. The intermediate centers are then extracted from each chunk. After that, the set of intermediate centers is again processed in order to generate the final cluster centers. The chunks are processed in the map phase while the intermediate centers are processed in the reduce phase. In the following, we first present the parallel implementation without considering MapReduce and then, we present how we have fitted the proposed solution using MapReduce framework.

To define the parallel implementation, it is necessary to define the algorithm that is applied on each chunk and the algorithm that is applied on the set of intermediate centers. For both phases, we use the KP+PS algorithm. For each chunk, the KP+PS algorithm is executed and k centers are extracted. Therefore, if we have p chunks, after applying KP+PS algorithm on each chunk, there will be a set of $k * p$ centers as the intermediate set. In order to obtain a good quality, we record the number of assigned data points to each extracted center. That is to say, we extract from each chunk, k centers and the number of data points assigned to each center. The number of assigned data points to each cluster center represents the importance of that center. Hence, we must extend the KP+PS algorithm to take into account the weighted data points when clustering the set of intermediate centers. In order to consider the weighted data points, we must change center update (3) and (4). If we take into account w_i as the weight of data point x_i, center of a final cluster must be calculated for numeric and categorical attributes using the following equations.

$$c_{jr} = \frac{\sum_{i=1}^{|c_j|} x_{ir} * w_i}{|c_j|}. \tag{6}$$

$$c_{jt} = a_t^h, \tag{7}$$

where

$$f(a_t^h) * w_i \geq f(a_t^z) * w_i, \quad \forall z, \quad 1 \leq z \leq m_c, \tag{8}$$

The parallel implementation of the KP+PS algorithm through MapReduce framework is straightforward. Each map task picks a chunk of data set, executes the KP+PS algorithm on that chunk and emits the extracted intermediate centers and their weights as the output. Once the map phase is finished, a set of intermediate weighted centers is collected as the output of the map phase, and this set of centers is emitted to a single reduce phase. The reduce phase takes the set of intermediate centers and their weights, executes again the KP+PS algorithm on them and returns the final centers as the output. Once the final cluster centers are generated, we assign each data point to the nearest cluster center.

Let X_i the chunk associated to map task i, p the number of chunks, $C^w = \left\{C_1^w \dots C_p^w\right\}$ the set of weighted intermediate centers where C_i^w the set of weighted intermediate centers extracted from chunk i and C^f the set of final centers. Algorithm 4 describes main steps of the proposed method.

Algorithm 4 The main steps of the proposed method

```
Input: $X$: Data set, $k$: number of clusters
Output: $C^f$: Final cluster centers
begin
    Divide the data set $X$ into a set of $p$ chunks X=$\{X_1 \dots X_p\}$.
    % Map Phase
    foreach $X_i \in X$ do
        Run the KP+PS algorithm on $X_i$ to get a set of $k$ weighted centers $C_i^w$.
        Emit $< 1/C_i^w >$ as the output of the map task.
    % Reduce Phase
    Run the KP+PS algorithm on $C^w$ to get a set of $k$ final centers $C^f$.
    Emit $< 1/C^f >$ as the output of the reduce phase.
end
```

4.3 Parameters selections

The proposed method needs three input parameters in addition to the number of clusters k. The first parameter is the chunk size, the second parameter is pruning heuristic $\alpha\%$ and the third parameter is the time to start the pruning strategy.

4.3.1 Tuning the chunk size

In theory, the minimum chunk size is k and the maximum chunk size is n. However, both of these extremes are not practical and value between k and n must be selected. According to Shahrivari and Jalili (2016), the most memory efficient value for chunk size is $\sqrt{k.n}$ because this value for chunk size generates a set of intermediate centers with size $k.n$. That is to say, the chunk size should not be set to value greater than $\sqrt{k.n}$. A smaller chunk size can yield to better quality since smaller chunk size produces more intermediate centers which represent the input data set. Hence, all experiments described in Section 5, assume that the chunk size is $\sqrt{k.n}$.

4.3.2 Tuning the pruning heuristic

The pruning heuristic $\alpha\%$ can be set between 1 to 100%. A small value of this parameter reduces significantly computational time since most of distance computations will be ignored leading to a small loss of clustering quality. Alternatively, setting the pruning heuristic to a large value does not reduce the high computational time with leading approximately the same partitioning of k-prototypes. The experimental results show that setting the pruning heuristic to 10% with respect to all of the pruning heuristics from 1 to 100% tested in this work gives a good result. Hence, we set the pruning heuristic to 10% to provide a good trade-off between efficiency and quality.

4.3.3 Tuning the time to start

As mentioned above, many of distance computations are redundant in k-prototypes method because data points usually stay in the same clusters after first iterations. When the pruning strategy starts from the first iteration, the proposed method may produces an inaccurate results. Otherwise when the pruning strategy starts from the last iteration, no many distance computations will be reduced. For this reason, the time to start the pruning strategy is extremely important. Hence, we start the pruning strategy from iteration 2 because the convergence speed of k-prototypes slows down substantially after iteration 2 in the sense that many data points remain in the same cluster.

4.4 Complexity analysis

A typical clustering algorithm has three types of complexities: time complexity, I/O complexity and space complexity. To give complexity analysis of AMRKP method, we use the following notations: n the number of data points, k the number of clusters, l the number of iterations, $\alpha\%$ the pruning heuristic, s the chunk size and p the number of chunks.

4.4.1 The time complexity analysis

The KP+PS algorithm requires at each iteration computing distances between centers and sorting them as shown in Algorithm 2. This step can be estimated by $O(k^2 + k^3) \cong O(k^3)$. Then, the KP+PS selects an $\alpha\%$ centers and evaluates the triangle inequality property until the property is not satisfied or all centers in the subset of $\alpha\%$ centers have been evaluated. Therefore, we can conclude that the KP+PS algorithm can reduce theoretically the time complexity of k-prototypes from $O(n.k.l)$ to $O((n.\alpha\%.k + k^3).l)$.

As stated before, the time to start the pruning strategy is extremely important. Hence, the time complexity of KP+PS algorithm depends on the iteration from which the pruning strategy is started. In the best case, when the pruning strategy is started from the first iteration, the time complexity of KP+PS is $O((n.\alpha\%.k+k^3).l)$. In the worst case, when KP+PS converges before the pruning strategy is started, then KP+PS falls back to k-prototypes, and the time complexity is $O((n.k+k^3).l)$. Therefore, we can conclude that the time complexity of KP+PS algorithm is bounded between $O((n.\alpha\%.k + k^3).l)$ and $O((n.k + k^3)l)$.

The KP+PS algorithm is applied twice: in the map phase and the reduce phases. In the map phase, each chunk involves running the KP+PS algorithm on that chunk. Therefore, the map phase takes $O(n/p.\alpha\%.k + k^3).l)$ time. In the reduce phase, the KP+PS algorithm must be executed on the set of intermediate centers which has $k.n/s$ data points. Hence, the reduce phase needs $O((n/s.\alpha\%.k + k^3).l)$ time. Given that $k.n/s << n/p$, the overall time complexity of the proposed method is $O(n/p.\alpha\%.k+k^3).l+(k.n/s.\alpha\%.k+k^3).l) \cong O(n/p.\alpha\%.k+k^3).l)$.

4.4.2 The I/O complexity analysis

The proposed method reads the input data set just one time from the file system in the map phase. Therefore, the I/O complexity of the map phase is $O(n/p)$. The I/O complexity of the reduce phase is $O(k.n/s)$. As a result, the overall I/O complexity of the proposed method is $O(n/p+k.n/s)$. If s is fixed to $\sqrt{k.n}$, the I/O complexity will be $O(n/p+\sqrt{k.n})$.

4.4.3 The space complexity analysis

The space complexity of AMRKP depends on the chunk size and the number of chunks that can be processed in the map phase. The map phase is required to keep p chunks in the memory. Hence, the space complexity of the map phase is $O(p.s)$. The reduce phase needs to store $k.n/s$ intermediate centers in the memory. Thus, the space complexity of the reduce phase is $O(k.n/s)$. As a result, the overall space complexity of the proposed method is evaluated by $O(p.s+k.n/s)$. If s is fixed to $\sqrt{k.n}$, the space complexity will be $O(p.s+\sqrt{k.n})$.

5 Experiments and results

5.1 Methodology

In order to evaluate the efficiency of AMRKP method, we performed experiments on both simulated and real data sets. We tried in this section to figure out three points. (i) How much is the efficiency of AMRKP method when applied to mixed large scale data compared to existing methods? (ii) How the pruning strategy can reduce the number of unnecessary distance computations? (iii) How the MapReduce framework can enhance the scalability of the proposed method when dealing with mixed large scale data?

We split experiments into three major subsections. First, we compare the performance of the proposed method versus the following existing methods: conventional k-prototypes, described in Algorithm 1, denoted by KP and the MapReduce-based k-prototypes (Ben Haj Kacem et al. 2015) denoted by MRKP. Then, we study the pruning strategy performance to reduce the number of distance computations between clusters and data points. Finally, we evaluate the MapReduce performance by analyzing the scalability of the proposed method.

5.2 Environment and data sets

The experiments are performed on a Hadoop cluster running the latest stable version of Hadoop 2.7.1. The Hadoop cluster consists of 6 machines. Each machine has 1-core 2.30 GHz CPU E5400 and 1GB of memory. The operating system of each machine is Ubuntu 14.10 server 64bit. We conducted the experiments on the following data sets:

- Simulated data set: four series of mixed large data sets are generated. The data sets range from 100 millions to 400 millions data points. Each data point is described using 5 numeric and 5 categorical attributes. The numeric values are generated with gaussian distribution. The mean is 350 and the sigma is 100. The categorical values are generated

Table 1 Summary of the data sets

Data set	Number of data points	Number of attributes	Domain
Sim100M	100.000.000	10 (5 Numeric, 5 Categorical)	Simulated
Sim200M	200.000.000	10 (5 Numeric, 5 Categorical)	Simulated
Sim300M	300.000.000	10 (5 Numeric, 5 Categorical)	Simulated
Sim400M	400.000.000	10 (5 Numeric, 5 Categorical)	Simulated
KDD	4.898.431	36 (33 Numeric, 3 Categorical)	Intrusion detection
Poker	1.000.000	10 (5 Numeric, 5 Categorical)	Game

using the data generator developed in.[1] In order to simplify the names of the simulated data sets, we will use the notations Sim100M, Sim200M, Sim300M and Sim400M to denote a simulated data set containing 100, 200, 300 and 400 millions data points respectively.

- KDD Cup data set (KDD): is a real data set which consists of normal and attack connections simulated in a military network environment. The KDD data set contains about 5 millions connections. Each connection is described using 33 numeric and 3 categorical attributes. The clustering process for this data set detects type of attacks among all connections. This data set was obtained from UCI machine learning repository.[2]
- Poker data set (Poker): is a real data set which is an example of a hand consisting of five playing cards drawn from a standard deck of 52 cards. The Poker data set contains about 1 millions data points. Each data point is described using 5 numeric attributes and 5 categorical attributes. The clustering process for this data set detects the hand situations. This data set was obtained from UCI machine learning repository.[3] Statistics of these data sets are summarized in Table 1.

5.3 Evaluation measures

In order to evaluate the quality of the proposed method, we use Sum Squared Error (SSE) (Xu and Wunsch 2010). It is one of the most common partitional clustering criteria which aims to measure the squared errors between each data point and the cluster center to which the data point belongs to. SSE can be defined by:

$$SSE = \sum_{i=1}^{n} \sum_{j=1}^{k} d(c_j, x_i), \tag{9}$$

where x_i the data point and c_j the cluster center.

In order to evaluate the ability of the proposed method to scale with large data sets, we use in our experiments the Speedup measure (Xu et al. 2002). It measures the ability of the designed parallel method to scale well when the number of machines increases and the size of data is fix. This measure is calculated as follows:

$$Speedup = \frac{T_1}{T_m}, \tag{10}$$

[1] https://projets.pasteur.fr/projects/rap-r/wiki/SyntheticDataGeneration.

[2] https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data.

[3] http://archive.ics.uci.edu/ml/datasets/Poker.

where T_1 the running time of processing data on 1 machine and T_m the running time of processing data on m machines in the used cluster.

To simplify the discussion of the experimental results in Tables 2, 3, 4 and 5, we will use the following conventions. Let ψ denotes either KP or MRKP. Let T, D and S denote respectively the running time, the distance computations and the quality of the clustering result in terms of SSE. Let β denotes either T, D or S. The enhancement of β_{AMRKP} (new algorithm) with respect to β_{ψ} (original algorithm) in percentage defined by:

$$\Delta_{\beta} = \frac{\beta_{AMRKP} - \beta_{\psi}}{\beta_{\psi}} * 100\%. \tag{11}$$

For example, the enhancement of the running time of AMRKP with respect to the running time of k-prototypes is defined by:

$$\Delta_T = \frac{T_{AMRKP} - T_{KP}}{T_{KP}} * 100\%, \tag{12}$$

where $\beta = T$ and $\psi = $ KP. It is important to note that as defined in (11), a more negative value of Δ_{β} implies a greater enhancement.

5.4 Comparison of the performance of AMRKP versus existing methods

We compare in this section the performance of the proposed method versus existing methods. The results are reported in Table 2. As the results show, AMRKP method always finishes several times faster than existing methods. For example, Table 2 shows that AMRKP method ($k = 10$ clusters) on Poker data set can reduce the running time by 88.62%

Table 2 Experimental results on simulated and real data sets

Data set	k	AMRKP-KP		AMRKP-MRKP	
		Δ_T	Δ_S	Δ_T	Δ_S
Sim100	10	−92.21	3.56	−45.05	3.56
	50	−92.14	4.78	−53.15	4.78
	100	−92.18	0.68	−53.23	0.68
Sim200	10	1.53	1.53	−45.42	1.53
	50	−90.85	−3.24	−43.53	−3.24
	100	−91.87	−2.25	−51.12	−2.25
Sim300	10	−90.86	4.65	−45.32	4.65
	50	−90.57	5.38	−43.86	5.38
	100	−92.10	−3.87	−52.75	−3.87
Sim400	10	−91.11	1.25	−46.69	1.25
	50	−90.86	3.86	−43.96	3.68
	100	−91.17	2.68	−50.94	2.68
KDD	10	−90.06	4.12	−53.66	4.12
	50	−83.66	5.68	−56.66	5.68
	100	−92.03	3.54	−59.92	3.54
Poker	10	−88.62	25.87	−59.32	25.87
	50	−89.17	25.75	−61.35	25.74
	100	−85.56	12.36	−62.55	12.36

Table 3 Comparison of the number of distance computations performed by AMRKP versus k-prototypes

Data set	k	AMRKP-KP	
		Δ_D	Δ_S
Sim100M	10	−16.56	3.25
	50	−29.85	5.44
	100	−55.19	6.53
Sim200M	10	−32.01	4.55
	50	−28.88	6.99
	100	−54.80	3.23
Sim300M	10	−15.89	4.55
	50	−22.13	3.17
	100	−55.12	2.86
Sim400M	10	−7.21	3.28
	50	−17.25	2.76
	100	−51.02	3.01
KDD	10	−62.11	5.31
	50	−77.51	4.17
	100	−85.56	4.55
Poker	10	−35.11	33.21
	50	−59.20	37.76
	100	−81.65	36.55

and by 59.32% compared to k-prototypes and MRKP respectively. In all of the data sets, more than 95% of the running time was spent in the map phase and this shows that AMRKP method is truly one pass. In addition, obtained results report that the proposed method converges to nearly same results of the conventional k-prototypes which allows to maintain a good quality of partitioning.

Table 4 The impact of the pruning heuristic $\alpha\%$ on the performance of AMRKP

Data set	Pruning heuristic (%)	AMRKP-KP	
		Δ_D	Δ_S
KDD	1	−88.56	36.55
	10	−85.85	36.44
	20	−84.19	34.53
	50	−80.17	30.50
	80	−76.88	6.99
	100	−74.66	0.01
Poker	1	−89.11	4.55
	10	−81.65	3.17
	20	−74.12	2.86
	50	−61.85	2.76
	80	−59.20	2.76
	100	−59.02	0.01

Table 5 The impact of the time to start on the performance of AMRKP

Data set	Time to start	AMRKP-KP	
		Δ_D	Δ_S
KDD	1	−89.11	236.55
	2	−76.33	5.93
	3	−48.05	3.65
	5	−45.22	3.05
	8	−19.27	2.75
	10	−9.61	0.08
Poker	1	−81.11	3.17
	2	−81.50	1.35
	3	−72.40	0.58
	5	−45.35	0.30
	8	−18.12	0.15
	10	−9.15	0.01

5.5 Pruning strategy performance

We evaluate in this section the performance of the pruning strategy to reduce the number of redundant distance computations. The results are reported in Table 3. As the results show, the proposed method reduces the number of distance computations over k-prototypes on both simulated and real data sets. We must also mention that this reduction becomes more significant with the increase of k. For example, the number of distance computations is reduced by 35.11% when $k = 10$ and by 81.65% when k =100 on Poker data set.

In another experiment we investigated the impact of pruning heuristic $\alpha\%$ on the performance of the proposed method compared to the conventional k-prototypes. For this purpose, we run AMRKP with different pruning heuristics from 1 to 100% on real data sets with $k =$ 100. The results are reported in Table 4. As shown in Table 4, when the pruning heuristic is fixed to a small value, many distance computations are reduced with small loss of quality. For example, when the pruning heuristic is 10%, the pruning strategy can reduce the number of distance computations of k-prototypes by 81.65% on Poker data set. But, when the pruning heuristic is fixed to a large value, a small number of distance computations are reduced while maintaining the clustering quality. For example, when the pruning heuristic is 80%, the pruning strategy can reduce the number of distance computations of k-prototypes by 59.20% on Poker data set.

Then, we investigated the impact of the time to start the pruning strategy on the performance of the proposed method compared to the conventional k-prototypes. For this purpose, we run AMRKP with different pruning'starting times from iteration 1 to iteration 10 on real data sets with $k = 100$. The results are reported in Table 5. As shown in Table 5, when the pruning strategy starts from the first iteration, AMRKP leads to significant reduction of distance computations and this may decrease the clustering quality. For example, when the pruning strategy starts from iteration 1, the AMRKP can reduce the number of distance computations of k-prototypes by 81.11% on Poker data set. On the other hand, when the pruning strategy starts from the last iteration, AMRKP leads to small reduction of distance computations with small loss of quality. For example, when the pruning strategy starts

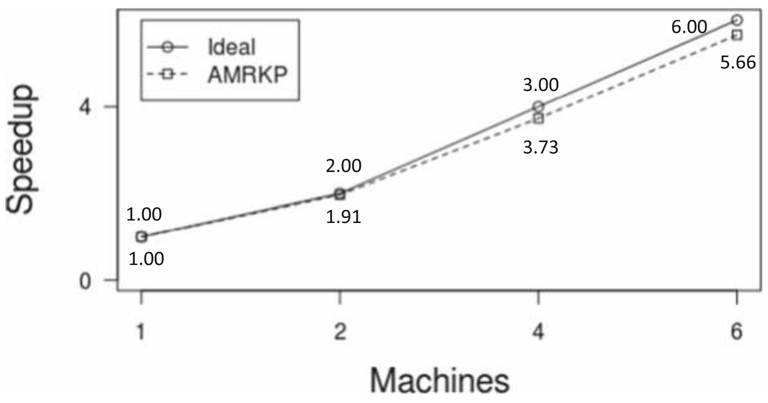

Fig. 2 Speedup of AMRKP as the number of machines increases

from iteration 10, the proposed method can reduce the number of distance computations of k-prototypes by 9.15% on Poker data set.

5.6 Scalability analysis

We first evaluate in this section the scalability of the proposed method when the number of machines increases. For investigating the speedup value, we used the Sim400M data set with $k = 100$. For computing speedup values, first we executed AMRKP method using just a single machine and then we added additional machines. Figure 2 illustrates the speedup results on Sim400M data set. The proposed method shows linear speedup as the number of machines increases because the MapReduce framework has linear speedup and each chunk can be processed independently.

Then, we evaluate the scalability of the proposed method when we increase the size of the data set. For investigating the influence of the size of data set, we used in this section the Sim100M, Sim200M, Sim300M and Sim400M data sets with $k = 100$. The results are plotted in Fig. 3. As the results show, the running time scales linearly, when size of data set

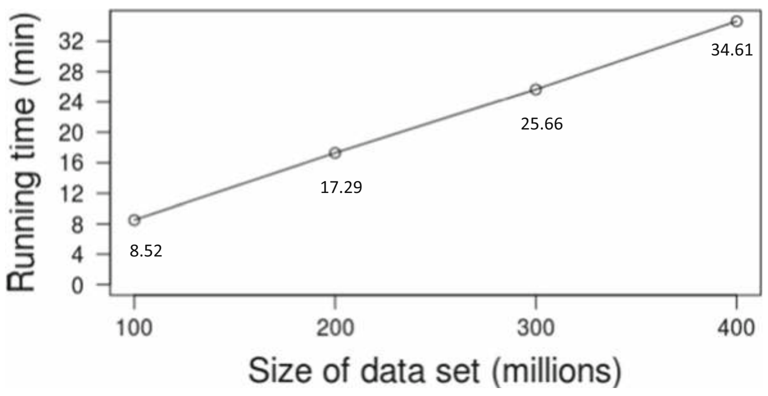

Fig. 3 Performance of AMRKP as the size of data set increases

increases. For example, the MRKP method takes more than one hour on Sim400M data set while the proposed method processed the data set in less than 40 minutes.

6 Conclusion

In order to deal with the issue of clustering mixed large scale data, we have proposed a new one-pass accelerated MapReduce-based k-prototypes clustering method. The proposed method reads and writes data only once which reduces largely the I/O operations like disk I/O and. Furthermore, the proposed method is based on a pruning strategy to accelerate the clustering process by reducing the redundant distance computations between cluster centers and data points. Experiments on huge simulated and real data sets have showed the efficiency of AMRKP to deal with mixed large scale data compared to existing methods.

The proposed method performs several iterations to converge to the optimal local solution. The number of iterations increases the running time while each iteration is time consuming. A good initialisation of the proposed method may improve both running time and clustering quality. An exciting direction for future works is to investigate the use of scalable initialisation techniques in order to reduce the number of iterations and then may be the improvement of the scalability of AMRKP method.

References

Ahmad, A., & Dey, L. (2007). A k-mean clustering algorithm for mixed numeric and categorical data. *Data Knowledge Engineering*, *63*(2), 503–527.

Ayech, M.W., & Ziou, D. (2015). Segmentation of Terahertz imaging using k-means clustering based on ranked set sampling. *Expert Systems with Applications*, *42*(6), 2959–2974.

Bahmani, B., Moseley, B., Vattani, A., Kumar, R., & Vassilvitskii, S. (2012). Scalable k-means++. *Proceedings of the VLDB Endowment*, *5*(7), 622–633.

Ben Haj Kacem, M.A., Ben N'cir, C.E., & Essoussi, N. (2015). MapReduce-based k-prototypes clustering method for big data. In *Proceedings of data science and advanced analytics* (pp. 1–7).

Ben HajKacem, M.A., N'cir, C.E., & Essoussi, N. (2016). An accelerated MapReduce-based K-prototypes for big data. In *Proceedings of software technologies: applications and foundations* (pp. 1–13).

Ben N'Cir, C.E., & Essoussi, N. (2015). Using sequences of words for non-disjoint grouping of documents. *International Journal of Pattern Recognition and Artificial Intelligence*, *29*(03), 1550013.

Dean, J., & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, *51*(1), 107–113.

Du, H., Wang, Y., & Dong, X. (2015). Texture image segmentation using affinity propagation and spectral clustering. *International Journal of Pattern Recognition and Artificial Intelligence*, *29*(05), 1555009.

Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.H., Qiu, J., & Fox, G. (2010). Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM international symposium on high performance distributed computing* (pp. 810–818). ACM.

Eliaz, K., & Spiegler, R. (2011). Consideration sets and competitive marketing. *The Review of Economic Studies*, *78*(1), 235–262.

Gandomi, A., & Haider, M. (2015). Beyond the hype: big data concepts, methods, and analytics. *International Journal of Information Management*, *35*(2), 137–144.

Gorodetsky, V. (2014). Big data: opportunities, challenges and solutions. In *Information and communication technologies in education, research, and industrial applications* (pp. 3–22).

Ji, J., Bai, T., Zhou, C., Ma, C., & Wang, Z. (2013). An improved k-prototypes clustering algorithm for mixed numeric and categorical data. *Neurocomputing*, *120*, 590–596.

Hadian, A., & Shahrivari, S. (2014). High performance parallel k-means clustering for disk-resident datasets on multi-core CPUs. *The Journal of Supercomputing*, *69*(2), 845–863.

Han, J., Pei, J., & Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

He, C., Chang, J., & Chen, X. (2010). Using the triangle inequality to accelerate TTSAS cluster algorithm. In *Electrical and control engineering (ICECE)* (pp. 2507–2510). IEEE.

Hussain, S.F., Mushtaq, M., & Halim, Z. (2014). Multi-view document clustering via ensemble method. *Journal of Intelligent Information Systems*, *43*(1), 81–99.

Huang, Z. (1997). 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).

Jain, A.K., Murty, M.N., & Flynn, P.J. (1999). Data clustering: a review. *ACM computing surveys (CSUR)*, *31*(3), 264–323.

Kim, Y., Shim, K., Kim, M.S., & Lee, J.S. (2014). DBCURE-MR: an efficient density-based clustering algorithm for large data using MapReduce. *Information Systems*, *42*, 15–35.

Lee, K.H., Lee, Y.J., Choi, H., Chung, Y.D., & Moon, B. (2012). Parallel data processing with MapReduce: a survey. *AcM sIGMoD Record*, *40*(4), 11–20.

Li, C., & Biswas, G. (2002). Unsupervised learning with mixed numeric and nominal data. *Knowledge and Data Engineering*, *14*(4), 673–690.

Liu, H.H., & Ong, C.S. (2008). Variable selection in clustering for marketing segmentation using genetic algorithms. *Expert Systems with Applications*, *34*(1), 502–510.

Ludwig, S.A. (2015). Mapreduce-based fuzzy c-means clustering algorithm: implementation and scalability. In *International journal of machine learning and cybernetics* (pp. 1–12).

Nanni, M. (2005). Speeding-up hierarchical agglomerative clustering in presence of expensive metrics. In *Pacific-Asia conference on knowledge discovery and data mining* (pp. 378–387). Berlin Heidelberg: Springer.

Ng, M.K., Li, M.J., Huang, J.Z., & He, Z. (2007). On the impact of dissimilarity measure in k-modes clustering algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(3), 503–507.

Shahrivari, S., & Jalili, S. (2016). Single-pass and linear-time k-means clustering based on MapReduce. *Information Systems*, *60*, 1–12.

Talbot, J., Yoo, R.M., & Kozyrakis, C. (2011). Phoenix++: modular MapReduce for shared-memory systems. In *Proceedings of the second international workshop on MapReduce and its applications* (pp. 9–16). ACM.

Tsai, C.F., Hsu, Y.F., Lin, C.Y., & Lin, W.Y. (2009). Intrusion detection by machine learning: A review. *Expert Systems with Applications*, *36*(10), 11994–12000.

Xu, R., & Wunsch, D.C. (2010). Clustering algorithms in biomedical research: a review. *Biomedical Engineering, IEEE Reviews*, *3*, 120–154.

Xu, X., Jeger, J., & Kriegel, H.P. (2002). A fast parallel clustering algorithm for large spatial databases. In *High performance data mining* (pp. 263–290).

Wang, G., Hao, J., Ma, J., & Huang, L. (2010). A new approach to intrusion detection using artificial neural networks and fuzzy clustering. *Expert Systems with Applications*, *37*(9), 6225–6232.

White, T. (2012). *Hadoop: the definitive guide*. O'Reilly Media Inc.

Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., & Stoica, I. (2010). Spark: cluster computing with working sets. *HotCloud*, *10*(10–10), 95.

Zhao, W., Ma, H., & He, Q. (2009). Parallel k-means clustering based on mapreduce. In *Proceedings of cloud computing* (pp 674–679).