Noname manuscript No.
(will be inserted by the editor)

MapReduce-based Fuzzy C-Means Clustering Algorithm: Implementation and Scalability

Simone A. Ludwig

Received: date / Accepted: date

Abstract The management and analysis of big data has been identified as one of the most important emerging needs in recent years. This is because of the sheer volume and increasing complexity of data being created or collected. Current clustering algorithms can not handle big data, and therefore, scalable solutions are necessary. Since fuzzy clustering algorithms have shown to outperform hard clustering approaches in terms of accuracy, this paper investigates the parallelization and scalability of a common and effective fuzzy clustering algorithm named Fuzzy C-Means (FCM) algorithm. The algorithm is parallelized using the MapReduce paradigm outlining how the *Map* and *Reduce* primitives are implemented. A validity analysis is conducted in order to show that the implementation works correctly achieving competitive purity results compared to state-of-the art clustering algorithms. Furthermore, a scalability analysis is conducted to demonstrate the performance of the parallel FCM implementation with increasing number of computing nodes used.

Keywords MapReduce · Hadoop · Scalability

1 Introduction

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

Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, ND, USA
E-mail: simone.ludwig@ndsu.edu

"As simulations and experiments yield ever more data, a fourth paradigm is emerging, consisting of the techniques and technologies needed to perform data intensive science". The question to address is how to effectively generate, manage and analyze the data and the resulting information. The solution requires a comprehensive, end-to-end approach that encompasses all stages from the initial data acquisition to its final analysis.

Data mining is is one of the most developed fields in the area of artificial intelligence and encompases a relatively broad field that deals with the automatic knowledge discovery from databases. Based on the rapid growth of data collected in various fields with their potential usefulness, this requires efficient tools to extract and utilize potentially gathered knowledge [2].

One of the important data mining tasks is classification, which is an effective method that is used in many different areas. The main idea behind classification is the construction of a model (classifier) that assigns items in a collection to target classes with the goal to accurately predict the target class for each item in the data [3]. There are many techniques that can be used to do classification such as decision trees, Bayes networks, genetic algorithms, genetic programming, particle swarm optimization, and many others [4]. Another important data mining technique that is used when analyzing data is clustering [5]. The aim of clustering algorithms is to divide a set of unlabeled data objects into different groups called clusters. The cluster membership measure is based on a similarity measure. In order to obtain high quality clusters, the similarity measure between the data objects in the same cluster should be maximized, and the similarity measure between the data objects from different groups should be minimized.

Clustering is the classification of objects into different groups, i.e., the partitioning of data into subsets (clusters), such that data in each subset shares some common features, often proximity according to some defined distance measure. Unlike conventional statistical methods, most clustering algorithms do not rely on the statistical distribution of data, and thus can be usefully applied in situations where little prior knowledge exists [6].

Most sequential classification/clustering algorithms suffer from the problem that they do not scale with larger sizes of data sets, and most of them are computationally expensive, both in terms of time and space. For these reasons, the parallelization of the data classification/clustering algorithms is paramount in order to deal with large scale data. To develop a good parallel classification/clustering algorithm that takes big data into consideration, the algorithm should be efficient, scalable and obtain high accuracy solutions.

In order to enable big data to be processed, the parallelization of data mining algorithms is paramount. Parallelization is a process where the computation is broken up into parallel tasks. The work done by each task, often called its grain size, can be as small as a single iteration in a parallel loop or as large as an entire procedure. When an application can be broken up into large parallel tasks, the application is called a coarse grain parallel application. Two common ways to partition computation are task partitioning, in which each task executes a certain function, and data partitioning, in which all tasks execute the same function but on different data.

This paper proposes the parallelization of a Fuzzy C-Means (FCM) clustering algorithm. The parallelization methodology used is the divide-and-conquer methodology referred to as MapReduce. The implementation details are explained in details outling how the FCM algorithm can be parallelized. Furthermore, a validity analysis is conducted in order to demonstrate the correct functioning of the implementation measuring the purity and comparing these to state-of-the art clustering algorithms. Moreover, a scalability analysis is conduced to investigate the performance of the parallel FCM implementation by measuring the speedup for increasing number of computing nodes used.

The remainder of this paper is as follows. Section 2 introduces clustering and fuzzy clustering in particular. The following section (Section 3) discusses related work in the area of big data processing. In Section 4, the implementation is described in details. The experimental setup and results are given in Section 5, and Section 6 concludes this work outlining the findings obtained.

2 Background to Clustering

Clustering can be applied to data that are quantitative (numerical), qualitative (categorical), or a mixture of both. The data usually are observations of some physical process. Each observation consists of n measured variables (features), grouped into an n-dimensional column vector $z_k = [z_{1k}, ..., z_{nk}]^T$, $z_k \in \mathbb{R}^n$. A set of N observations is denoted by $Z = \{z_k \mid k = 1, 2, ..., N\}$, and is represented as an $n \times N$ matrix [6]:

$$Z = \begin{pmatrix} z_{11} & z_{12} & ... & z_{1N} \\ z_{21} & z_{22} & ... & z_{2N} \\ ... & ... & ... & ... \\ z_{n1} & z_{n2} & ... & z_{nN} \end{pmatrix} \tag{1}$$

There are several definitions of how a cluster can be formulated depending on the objective of clustering. In general, a cluster is a group of objects that are more similar to one another than to members of other clusters [7, 8]. The term “similarity” is defined as mathematical similarity and can be defined by a distance norm. Distance can be measured among the data vectors themselves or as a distance from a data vector to some prototypical object (prototype) of the cluster. Since the prototypes are usually not known beforehand, they are determined by the clustering algorithms simultaneously with the partitioning of the data. The prototypes can be vectors of the same dimension as the data objects, however, they can also be defined as “higher-level” geometrical objects such as linear or nonlinear subspaces or functions. The performance of most clustering algorithms is influenced by the geometrical shapes and densities of the individual clusters, but also by spatial relations and distances among the clusters. Clusters can be well-separated, continuously connected or overlapping [6].

Many clustering algorithms have been introduced and clustering techniques can be categorized depending on whether the subsets of the resulting classification are *fuzzy* or *crisp* (hard). *Hard clustering* methods are based on classical set theory and require that an object either does or does not belong to a cluster. Hard clustering means that the data is partitioned into a specified number of mutually exclusive subsets. *Fuzzy clustering* methods, however, allow the objects to belong to several clusters simultaneously with different degrees of membership. Fuzzy clustering is seen as more natural than hard clustering since the objects on the boundaries between several classes are not forced to fully belong to one of the classes, but rather are assigned membership degrees between 0 and 1 indicating their partial membership. The concept of *fuzzy partition* is vital for cluster analysis, and therefore, also for the identification techniques that are based on fuzzy clustering. Fuzzy and possibilistic partitions are seen as a generalization of *hard partition* which is formulated in terms of classical subsets [6].

The objective of clustering is to partition the data set Z into c clusters. For now let us assume that c is known based on prior knowledge. Using classical sets, a *hard partition* of Z can be defined as a family of subsets A_i $1 \leq i \leq c \subset P(Z)^1$ with the following properties [6, 7]:

$$\bigcup_{i=1}^{c} A_i = Z \tag{2a}$$

$$A_i \cap A_j = \emptyset, 1 \leq i \neq j \leq c \tag{2b}$$

$$\emptyset \subset A_i \subset Z, 1 \leq i \leq c \tag{2c}$$

Equation 2a denotes that the union subset A_i contains all the data. The subsets have to be disjoint as stated by Equation 2b, and none of them can be empty nor contain all the data in Z as given by Equation 2c. In terms of *membership (characteristic) function*, a partition can be conveniently represented by the *partition matrix* $U = [\mu_{ik}]_{c \times N}$. The ith subset A_i of Z. It follows from Equations 2 that the elements of U must satisfy the following conditions [6]:

$$\mu_{ik} \in \{0, 1\}, 1 \leq i \leq c, 1 \leq k \leq N \tag{2d}$$

$$\sum_{i=1}^{c} \mu_{ik} = 1, 1 \leq k \leq N \tag{2e}$$

$$0 < \sum_{k=1}^{N} \mu_{ik} < N, 1 \leq i \leq c \tag{2f}$$

Thus, the space of all possible hard partition matrices for Z referred to as the hard partitioning space is defined by:

$$M_{HC} = \left\{ U \in \mathbb{R}^{c \times N} | \mu_{ik} \in \{0, 1\}, \forall i, k; \sum_{i=1}^{c} \mu_{ik} = 1, \forall k; 0 < \sum_{k=1}^{N} \mu_{ik} < N, \forall i \right\} \tag{2g}$$

Generalization of the hard partitioning to the fuzzy partitioning follows by allowing μ_{ik} to obtain values in $[0, 1]$. The conditions for a fuzzy partition matrix, analogous to hard clustering equations, is given by [6, 9]:

$$\mu_{ik} \in [0, 1], 1 \leq i \leq c, 1 \leq k \leq N \tag{3a}$$

$$\sum_{i=1}^{c} \mu_{ik} = 1, 1 \leq k \leq N \tag{3b}$$

$$0 < \sum_{k=1}^{N} \mu_{ik} < N, 1 \leq i \leq c \tag{3c}$$

The ith row of the fuzzy partition matrix U contains values of the ith *membership function* of the fuzzy subset A_i of Z. Equation 3b constrains the sum of each column to 1, and thus the total membership of each z_k in Z equals one. The fuzzy partitioning space for Z is the set [6]:

$$M_{FC} = \left\{ U \in \mathbb{R}^{c \times N} | \mu_{ik} \in [0, 1], \forall i, k; \sum_{i=1}^{c} \mu_{ik} = 1, \forall k; 0 < \sum_{k=1}^{N} \mu_{ik} < N, \forall i \right\} \tag{3d}$$

3 Related Work

In this section, first related work in the area of clustering is introduced, and afterwards clustering techniques that make use of parallelization mechanisms applied to big data are described. Given that this paper is concerned with the implementation and evaluation of a parallelized FCM algorithm, the related work outlines other research conducted in this area.

Fuzzy clustering can be categorized into three categories, namely hierarchical fuzzy clustering methods, graph-theoretic fuzzy clustering methods, and fuzzy clustering [10].

Hierarchical clustering techniques generate a hierarchy of partitions by means of agglomerative and divisive methods [10]; whereby the agglomerative algorithms produce a sequence of clusters of decreasing number by merging two clusters from the previous level. The divisive algorithms on the other hand perform the clustering the other way around. In [11] a hierarchical clustering algorithm in the area of business systems planning is proposed. The best cluster count is determined by a matching approach. In another technique, called fuzzy equivalent relation-based hierarchical clustering, the clustering is managed without needing a predefined count of clusters [12].

Another category referred to as graph-theoretic fuzzy clustering methods is based on the notion of connectivity of nodes of a graph representing the data set. In particular, the graph representing the data structure is a fuzzy graph and different types of connectivity leading to different types of clusters. The fuzzy graph idea was first mentioned in [13], whereby the fuzzy analogues of several basic graph-theoretic concepts such as bridges, cycles, paths, trees were introduced. Fuzzy graphs were first applied to cluster analysis in [14].

Fuzzy clustering that used different objective functions has shown to give better formulation to clustering. The heavily used fuzzy C-Means clustering model (FCM) was first introduced in 1974 [15], and was later extended and generalized in [7]. Many variations of the method and model improvements have been suggested since then.

The Gustafson-Kessel (GK) algorithm [16] is a fuzzy clustering method which estimates the local covariance by the partitioning of the data into subsets that can be fitted with linear submodels. Nevertheless, considering a general structure for the covariance matrix can have a substantial effect on the modeling approach, and thus, the Gath-Geva algorithm [17] was proposed to overcome the fitting problem. A fuzzy clustering algorithm where the prototype of each cluster is multi-dimensional linear was proposed by the Fuzzy C-Varieties (FCV) [18] clustering algorithm.

Replacing the Euclidean distances with other distance measures and enriching the cluster prototypes with further parameters allows for other shapes than only spherical clusters to be discovered. Clusters might be ellipsoidal, linear, manifolds, quadrics or have even different volumes [10]. The main benefit of fuzzy clustering has been proven to handle ambiguous data that share properties of different clusters by the notion of different membership degrees to be assigned to data objects.

The use of fuzzy clustering algorithms, in particular the FCM algorithm, has been successfully applied to the area of digital image processing and image segmentation [19] in particular, where the technique showed to be very efficient. However, the drawback the FCM algorithm still exhibits is the robustness to noise and outliers, especially in absence of prior knowledge of noise. A crucial parameter, used to balance between robustness to noise and effectiveness of preserving the details of the image, is manually set based on experience. In addition, the time of segmenting an image depends on the image size, and hence the larger the size of the image, the longer the segmentation time [20].

In [21], a K-means clustering variant is proposed. The approach is based on a prototype-based hybrid approach that speeds-up the k-means clustering method. The method first partitions the data set into small clusters (grouplets). Each grouplet is first represented by a prototype, and then the set of prototypes is partitioned into k clusters using the modified k-means method. The modified k-means method avoids empty clusters. Each prototype is replaced by its corresponding set of patterns to derive a partition of the data set. The proposed method has shown to be much faster in terms of processing time than basic K-means.

Big data clustering has recently received significant amount of attention. In particular, the aim is to build efficient and effective parallel clustering algorithms. Many of these algorithms use the MapReduce methodology that was proposed by Google [22]. In the following we will review research that has used and applied the MapReduce paradigm to the clustering of big data sets.

A MapReduce design and implementation of an efficient DBSCAN algorithm is introduced in [23]. The proposed algorithm addresses the drawbacks of existing parallel DBSCAN algorithms such as the data balancing and scalability issues. The algorithm tackles these issues using a parallelized implementation that removes the sequential processing bottleneck and thereby improves the algorithm's scalability. Furthermore, the algorithm showed the largest improvement when the data was imbalanced. The evaluation conducted using large scale data sets demonstrate the efficiency and scalability of their algorithm.

A parallel K-means clustering algorithm based on MapReduce was proposed in [24]. The algorithm locates the centroids by calculating the weighted average of each individual cluster points via the *Map* function. The *Reduce* function then assigns a new centroid to each data point based on the distance calculations. At the end, a MapReduce iterative refinement technique is applied to locate the final centroids. The authors evaluated the implementation using the measures of speedup, scaleup, and sizeup. The results showed that the proposed algorithm is able to process large data sets of 8 GB using commodity hardware effectively.

In [25], an algorithm for solving the problem of document clustering using the MapReduce-based K-means algorithm is proposed. The algorithm uses the MapReduce paradigm to iterate over the document collection in order to calculate the term frequency and inverse document frequency. The algorithm represents the documents as <key,value> pairs, where the key is the document type and the value is the document text. The authors compare a non-parallelized version of K-means with the parallelized version to show the speedup gain. Furthermore, the experiments of the parallelized k-means algorithm on 50,000 documents showed that the algorithm performs very well in terms of accuracy for this text clustering task while having a reasonable execution time.

Another K-means clustering algorithm using MapReduce by merging the K-means algorithm with the ensemble learning bagging method is introduced in [26]. The proposed algorithm addresses the instability and sensitivity to outliers. Ensemble learning is a technique that uses a collection of models to achieve better results than any model in the collection, and bagging is one of the most popular type of ensemble techniques. The evaluation was performed on relatively small data sets (instances around 5,000) and only 4 nodes were used for the parallelization. The authors show that a speedup is obtained on data sets consisting of outliers.

A Self-Organizing Map (SOM) was modified to work with large scale data sets by implementing the algorithm using the MapReduce concept to improve the performance of clustering as shown in [27]. A self-organizing map is an unsupervised neural network that projects high-dimensional data onto a low-dimensional grid and visually represents the topological order of the original data. Unfortunately, the details of the MapReduce implementation are not given, but the experiments that were conducted with a small data set demonstrated the efficiency of the MapReduce-based SOM.

In [28], the authors applied the MapReduce framework to solve co-clustering problems by introducing a framework called DISCO (DIStributed CO-clustering with MapReduce). Unlike clustering which groups similar rows or columns independently, co-clustering searches for interrelated submatrices of rows and columns. The authors proved that using MapReduce is a good solution for co-clustering mining tasks by applying the algorithm to data sets such as collaborative filtering, text mining, etc. The experiments demonstrated that co-clustering with MapReduce can scale well with large data sets using up to 40 nodes.

In [29], a fast clustering algorithm with a constant factor approximation guarantee was proposed. The authors use a sampling technique to reduce the data size first, and then apply the Lloyd's algorithm on the remaining data set. A comparison of this algorithm with several sequential and parallel algorithms for the k-median problem was conducted using randomly generated data sets to evaluate the performance of the algorithm. The randomly generated data sets contained up to 10 million points. The results showed that the algorithm achieves better or similar solutions compared to the existing algorithms especially on very large data sets.

A big data clustering method based on the MapReduce framework was proposed in [30]. The authors used an ant colony approach to decompose the big data into several data partitions to be used in parallel clustering. Applying MapReduce to the ant colony clustering algorithm lead to the automation of the semantic clustering to improve the data analysis task. The proposed algorithm was developed and tested on data sets with large number of records (up to 800K) and showed acceptable accuracy with good speedup.

In [31], the authors introduced a new approach, called the Best Of both Worlds (BOW) method to minimize the I/O cost of cluster analysis with the MapReduce model by minimizing the network overhead among the processing nodes. They proposed a subspace clustering method to handle very large data sets in a reasonable amount of time. Experiments on terabyte data sets were conducted using 700 mappers and up to 140 reducers. The results showed very good speedup results.

As can been seen by the related work, different parallel clustering methods have been proposed in the past. The aim of this paper is to parallelize the FCM algorithm using the MapReduce concept in order to conduct a thorough experimentation and scalability analysis.

4 MapReduce-based Fuzzy C-Means Implementation

4.1 Fuzzy C-Means Algorithm

Most analytical fuzzy clustering algorithms are based on the optimization of the basic c-means objective function, or some modification of the objective function. The optimization of the c-means functional represents a nonlinear minimization problem, which can be solved by using a variety of methods including iterative minimization. The most popular method is to use the simple Picard iteration through the first-order conditions for stationary points, known as the Fuzzy C-Means (FCM) algorithm. Bezdek has proven the convergence of the FCM algorithm in [7]. An optimal c partition is produced iteratively by minimizing the weighted within group sum of squared error objective function:

$$J = \sum_{i=1}^{n} \sum_{j=1}^{c} (u_{ij})^m d^2(y_i, c_j) \tag{4}$$

where $Y = [y_1, y_2, ..., y_n]$ is the data set in a d-dimensional vector space, n is the number of data items, c is the number of clusters which is defined by the user where $2 \leq c \leq n$, u_{ij} is the degree of membership of y_i in the j^{th} cluster, m is a weighted exponent on each fuzzy membership, c_j is the center of cluster j, $d^2(y_i, c_j)$ is a square distance measure between object y_i and cluster c_j. An optimal solution with c partitions can be obtained via an iterative process described in Algorithm 1. First, the fuzzy partition matrix is initialized, then within an iterative loop the following steps are executed: the cluster centers are updated using Equation 5, then the membership matrix is updated using Equations 6 and 7. The loop is repeated until a certain threshold value is reached [32].

Algorithm 1 FCM Algorithm

```
Input: $c$: centroid matrix, $m$: weighted exponent of fuzzy membership, $\epsilon$: threshold value used as stopping criterion, $Y = [y_1, y_2, ..., y_n]$: data
Output: $c$: updated centroid matrix

Randomly initialize the fuzzy partition matrix $U = [u_{ij}^k]$
repeat
    Calculate the cluster centers with $U^k$:
        $c_j = \frac{\sum_{i=1}^{n} (u_{ij}^k)^m y_i}{\sum_{i=1}^{n} (u_{ij}^k)^m}$    (5)
    Update the membership matrix $U^{k+1}$ using:
        $u_{ij}^{k+1} = \frac{1}{\sum_{k=1}^{c} (\frac{d_{ij}}{d_{kj}})^{\frac{2}{(m-1)}}}$    (6)
    where
        $d_{ij} = ||y_i - c_j||^2$    (7)
until $max_{ij} ||u_{ij}^k - u_{ij}^{k+1}|| < \epsilon$
Return $c$
```

4.2 MapReduce-based Fuzzy C-Means Algorithm (MR-FCM)

The growth of the internet has challenged researchers to develop new ideas to deal with the ever increasing amount of data. Parallelization of algorithms is needed in order to enable big data processing. Classic parallel applications that were developed in the past either used message passing runtimes such as MPI (Message Passing Interface) [33] or PVM (Parallel Virtual Machines) [34].

A parallel implementation of the fuzzy c-means algorithm with MPI was proposed in [35]. The implementation consists of three Master/Slave processes, whereby the first computes the centroids, the second computes the distances and updates the partition matrix as well as updates the new centroids, and the third calculates the validity index. Moderately sized data sets were used to evaluate the approach and good speedup results were achieved.

However, MPI utilizes a rich set of communication and synchronization constructs, which need to be explicitly programmed. In order to make the development of parallel applications easier, Google introduced a programming paradigm called MapReduce that uses the *Map* and *Reduce* primitives that are present in functional programming languages. The MapReduce implementation enables large computations to be divided into several independent *Map* functions. MapReduce provides fault tolerance since it has a mechanism that automatically re-executes *Map* or *Reduce* tasks that have failed.

The MapReduce model works as follows. The input of the computation is a set of key-value pairs, and the output is a set of output key-value pairs. The algorithm to be parallelized needs to be expressed by *Map* and *Reduce* functions. The *Map* function takes an input pair and returns a set of intermediate key-value pairs. The framework then groups all intermediate values associated with the same intermediate key and passes them to the *Reduce* function. The *Reduce* function uses the intermediate key and set of values for that key. These values are merged together to form a smaller set of values. The intermediate values are forwarded to the *Reduce* function via an iterator. More formally, the *Map* and *Reduce* functions have the following types:

$map(k_1, v_1) \rightarrow list(k_2, v_2)$

$reduce(k_2, list(v_2) \rightarrow list(v_3)$

In order to consider the mapping of the FCM algorithm to the *Map* and *Reduce* primitives, it is necessary for FCM to be partitioned into two MapReduce jobs since only one would not be sufficient. The first MapReduce job calculates the centroid matrix by iterating over the data records, and a second MapReduce job is necessary since the following calculations need the complete centroid matrix as the input. The second MapReduce job also iterates over the data records and calculates the distances to be used to update the membership matrix as well as to calculate the fitness.

The details of the proposed implementation in the form of block diagrams are given in Figures 1 and 2. As can be seen, in the first MapReduce job the mappers have a portion of the data set and a portion of the membership matrix

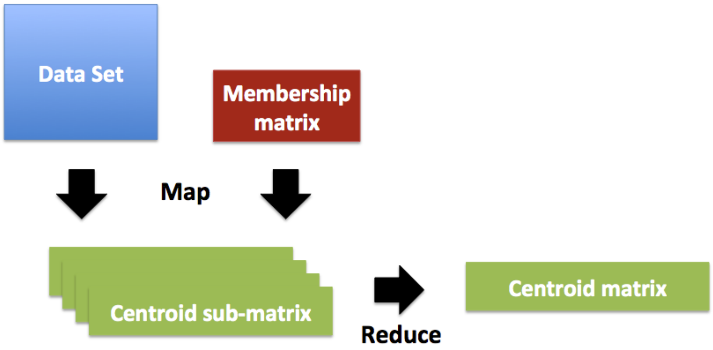


Fig. 1 First MapReduce job

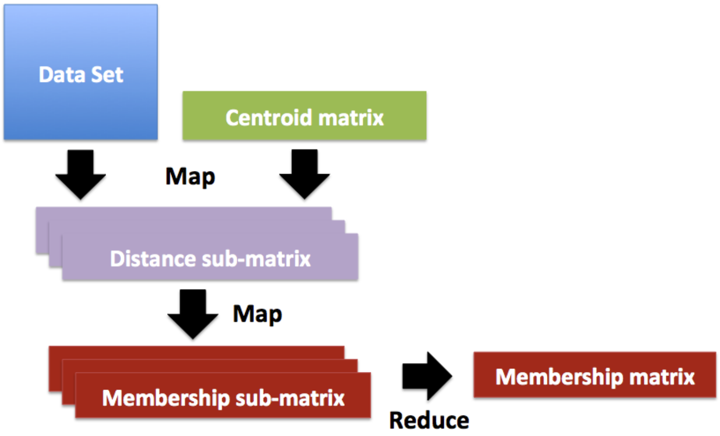


Fig. 2 Second MapReduce job

and produce centroid sub-matrices. The reducer of the first MapReduce job then merges the sub-matrices into the centroid matrix.

The second MapReduce job compared to the first involves more computations to be executed. During the *Map* phase, portions of the data set are received and the distance sub-matrices and membership matrices, and sub-objective values are computed. Again, in the *Reduce* phase the membership sub-matrices are merged, and the sub-objective values are summed up. Please note that even though the figure shows three *Map* arrows, however, these steps are done in one *Map* phase.

An algorithmic description detailing the Main procedure and the two MapReduce jobs is given in Algorithm 2-6. The Algorithm 2 proceeds as follows. First, the membership matrix is randomly initialized. Then, the data set needs to be prepared such that the data set itself and the membership matrix are merged together vertically in order for the first MapReduce job to have the data (both the data set as well as the membership matrix) available. This data set is stored in HDFS to be ready for the first MapReduce job. After this, the

Hadoop framework calls the first and then the second MapReduce job. The resulting updated membership matrix is copied from the HDFS to the local file system for the second iteration to begin. The algorithm iterates calling the two mapreduce jobs several times until the stopping condition is met. Once this is completed the purity value is calculated using Equation 8.

Algorithm 2 Main Procedure of MR-FCM Algorithm

```
Input: data set
Output: purity

randomly initialize membership matrix
while stopping condition is not met do
    vertically merge data set with membership matrix and store in HDFS
    Hadoop calls map and reduce jobs of first and then second mapreduce operation
    copy membership matrix from HDFS and store locally
end while
calculate purity (Equation 8) and write result to file
```

Algorithm 3 shows the pseudo code for the *Map* function of the first MapReduce job. The inputs are the data record values and the membership matrix (which are vertically merged), and the output is the intermediate centroid matrix. During this *Map* function the intermediate centroid matrix values are calculated using Equation 5, which are then emitted.

Algorithm 3 Map(key,value) of MapReduce job 1

```
Input: key: data record, value: data record values and membership matrix
Output: <key',value'> pair, where value' is the intermediate centroid matrix

for each key do
    calculate intermediate centroid matrix using Equation 5 and store in value'
    emit <key',value'> pair
end for
```

The intermediate centroid matrices are then merged by the *Reduce* function of the first MapReduce job 1 as shown in Algorithm 4.

Algorithm 4 Reduce(key,value) of MapReduce job 1

```
Input: key: data record, value: intermediate centroid values
Output: <key',value'> pair, where value' is the centroid values

for each key do
    calculate centroid values by summing over intermediate centroid values and store in
value'
    emit <key',value'> pair
end for
```

Algorithm 5 shows the *Map* function of the second MapReduce job. The function takes as the inputs the data record values and the centroid matrix. The output is an intermediate membership matrix. This *Map* function first calculates the distances between the data points and the centroids using Equation 7, and then updates the intermediate membership matrix using Equation 6, which is then emitted.

Algorithm 5 Map(key,value) of MapReduce job 2

```
Input: key: data record, value: data record values and centroid matrix
Output: <key',value'> pair, where value' is the intermediate membership matrix

for each key do
    calculate distances using Equation 7
    update intermediate membership matrix using Equation 6 and store in value'
    emit <key',value'> pair
end for
```

The *Reduce* function then merges the intermediate membership matrices and emits the membership matrix to be used in the next iteration. Algorithm 6 shows the details.

Algorithm 6 Reduce(key,value) of MapReduce job 2

```
Input: key: data record, value: intermediate membership matrix
Output: <key',value'> pair, where value' is the membership matrix

for each key do
    merge intermediate membership matrices and store in value'
    emit <key',value'> pair
end for
```

5 Experiments and Results

5.1 Clustering Data Set

The cover type data [36] set was used as the base for the experimentation. The data set contains data regarding 30 x 30 meter patches of forest such as elevation, distance to roads and water, hillshade, etc. The data set is used to identify which type of cover a particular patch of forest belongs to, and there are seven different types defined as output. The data set provides useful information to natural resource managers in order to allow them to make appropriate decisions about ecosystem management strategies. The data set characteristics are the following: number of instances is 581,012, number of attributes (categorical, numerical) is 54, and number of classes/labels is 7.

5.2 Computing Environment

Most of the experiments were conducted on the Longhorn Hadoop cluster hosted by the Texas Advanced Computing Center (TACC)[1] consisting of 48 nodes containing 48GB of RAM, 8 Intel Nehalem cores (2.5GHz each), which results in 384 compute cores with 2.304 TB of aggregated memory. For the experiments with the Mahout algorithms, another cluster, the Rustler Hadoop cluster[2] was used. The Rustler cluster consists of 66 nodes computing cluster for data intensive computing. Each node has dual eight core Ivy Bridge CPUs (Intel(R) Xeon(R) CPU E5-2650) running at 2.60GHz, with 128 GB DDR3 memory and 16 1TB SATA drives providing the backing for the HDFS file system. All nodes are connected via a 10 Gbps network. Hadoop version 0.21 was used for the MapReduce framework, and Java runtime 1.7 for the system implementation. The Apache Hadoop software library is a framework allowing distributed processing of large data sets across clusters of computers using simple programming models such as MapReduce [37]. Hadoop consists of MapReduce (processing element), and the Hadoop Distributed File System (HDFS) (storage element).

5.3 Validation of the MR-FCM algorithm

In order to guarantee that the parallelized version of the FCM algorithm works appropriately, a performance measure needs to be used to quantify the resulting clustering quality. Given that the cover type data set is a “classification data set”, which includes the correct labels, therefore, the clustering quality can be measured in terms of purity [38]. Purity is defined as:

$$Purity = \frac{1}{n} \sum_{j=1}^{k} \max_i (\mid L_i \cap C_j \mid) \tag{8}$$

where C_j contains all data instances assigned to cluster j by the clustering algorithm, n is the number of data instances in the data set, k is the number of clusters that is generated from the clustering process, L_i denotes the true assignments of the data instances to cluster i.

In order to compare the MR-FCM algorithm, the following comparison algorithms are used: CLUTO (CLUstering TOolkit) [39], approximate kernel Possibilistic C-Means (akPCM) [40], and approximate kernel Fuzzy C-Means (akFCM) [40]. The difference between the different algorithms are the membership update equations. CLUTO uses the hard c-means membership update, akPCM uses the possibilistic c-means membership update, and akFCM uses the fuzzy c-means membership update, which our algorithm also uses. More details on the membership update equations can be found in [41]. In addition,

[1] https://portal.longhorn.tacc.utexas.edu/

[2] https://www.tacc.utexas.edu/systems/rustler

Table 1 Purity results for different clustering algorithms

Algorithm	Purity scores
CLUTO	0.49
KMeans	0.50
akPCM	(RBF) 0.50
	(D2) 0.49
akFCM	(RBF) 0.52
	(D2) 0.53
FKM	0.52
MR-FCM	0.52

the KMeans as well as a recently released Fuzzy k-Means (FKM) algorithm (released in 2014) from the Mahout library [42] were also used for comparison.

The parameters for the proposed MR-FCM algorithm as well as for the FKM algorithm were set as follows:

- cluster number = 7
- fuzzification exponent = 1.5
- epsilon = 1.0E-5

Table 1 lists the purity values for the different algorithms. For akPCM and akFCM the distinction is made using different kernels: Radial Basis Functions abbreviated as RBF, and Degree-2 Polynominal abbreviated as D2. We can see that our MR-FCM implementation achieves comparable purity results compared to akFCM and FKM, and better results than CLUTO, KMeans and akPCM.

5.4 Scalability Results

Since different data set sizes are necessary for the performance evaluation, the cover type data set has been split into different portions as shown in Table 2. Listed is the name of the data set, which corresponds to the number of rows contained, as well as the file size is given in bytes. Since the number of rows of the complete cover type data set is 581,012, for the experiments of the larger data set sizes, rows of the data sets were duplicated.

Three different experiments were conducted. The first experiment compares the ratio of the number of mappers and reducers used, and the second experiment uses the findings from the first experiment to perform a scalability analysis of the number of mappers/reducers used. The third experiments compares the Mahout FKM implementation with the proposed MR-FCM algorithm.

The difference between the two sets of measurements is that for the first experiment the same number of mappers and reducers are used for the second MapReduce job.

For the first MapReduce job preliminary experiments showed that the best performance is achieved when the number of reducers and the number of map-

Table 2 Data set description for performance evaluation

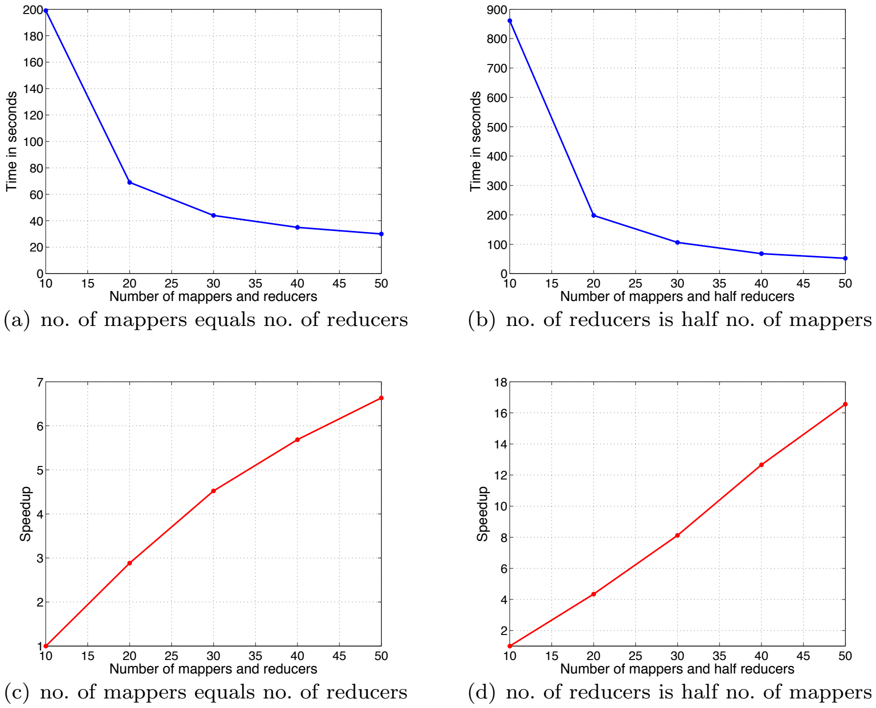
Data set	File size in bytes
100K	18,237,388
200K	36,606,436
300K	55,011,262
400K	73,387,535
500K	91,761,302
600K	109,998,690
700K	128,367,738
800K	146,772,564
900K	165,148,837
1M	183,522,604
2M	367,045,208
3M	550,567,812
4M	734,090,416
5M	917,613,020

pers is 7, since there are 7 outcomes (clusters) in the data set. Therefore, in all experiments conducted the mappers and reducers for the first MapReduce job were set to 7, however, the timings of the first MapReduce (MR1) job are also be reported.

Figure 3 shows the execution time and the speedup of the second MapReduce (MR2) job. The left hand side (Figures 3(a) and 3(c)) shows the results of using equal number of mappers and reducers, and the right hand side (Figures 3(b) and 3(d)) shows the results of using half the number of mappers for the reducers. The speedup results are calculated based on the time in 10 mappers and 10 reducers, and 10 mappers and 5 reducers, as shown in Figures 3(c) and 3(d), respectively. What can be seen from this comparison is that the utilization of the Hadoop framework is much better when only half the number of mappers are used for the reducers. In particular, using 10 mappers and 10 reducers, the MR2 time is 861 seconds, whereas the MR2 time is 199 seconds when 10 mappers and only 5 reducers are used. Therefore, an improvement of factor 4 is achieved with the setup of number of reducers equals half of the number of mappers. This finding is used throughout the next set of experiments.

The second experiment conducts a scalability analysis whereby the data set sizes are increased by 100,000 rows starting with a data set of 100,000 rows all the way up to 900,000 rows. Again, the number of mappers and reducers for the MR1 job was set to 7. Table 3 shows the execution time of MR1 for varying data set sizes. We observe a normal linear trend of the execution time for increasing data set sizes given that the number of mappers and reducers is fixed to 7.

Figure 4 shows the execution time of MR2 for increasing numbers of mapper and reducers used. The experiments use increments of 50, starting with 50 mappers (25 reducers) and ending with 500 mappers (250 reducers). Looking at all the figures we can see that the first figures with the lower numbers of mappers used show an “exponential” increase, whereas the later figures with



(a) no. of mappers equals no. of reducers

(b) no. of reducers is half no. of mappers

(c) no. of mappers equals no. of reducers

(d) no. of reducers is half no. of mappers

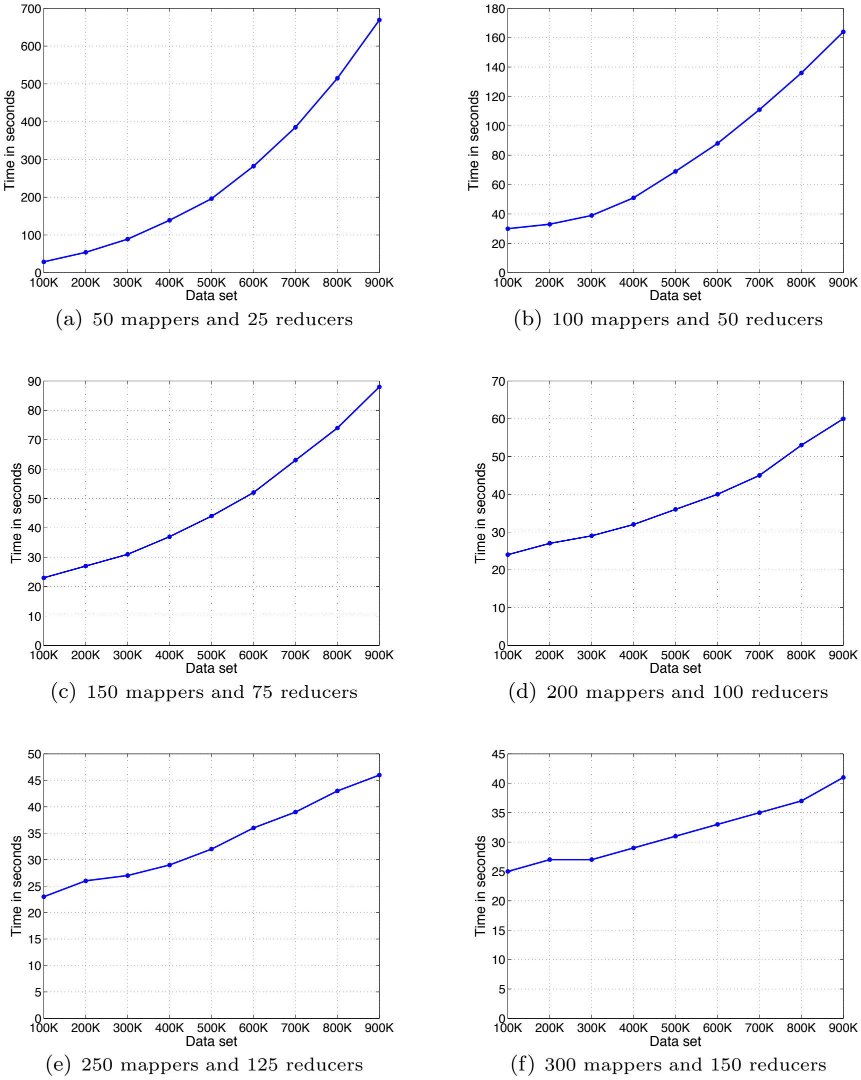
Fig. 3 Time in seconds and speedup results for MR2

Table 3 Time in seconds for MR1 for varying data set sizes (no. of mappers and reducers equals 7)

Data set	MR1 time (seconds)
100K	86 ± 3
200K	156 ± 1
300K	217 ± 2
400K	285 ± 2
500K	377 ± 11
600K	429 ± 8
700K	493 ± 9
800K	553 ± 14
900K	628 ± 9

the higher numbers of mappers used “flatten” and show an almost linear increase. The execution times for the 900K data set are 669, 164, 88, 60, 46, and 41 seconds for 50, 100, 150, 200, 250, and 300 mappers, respectively.

The third experiment compares the two Mahout algorithms KMeans and FKM with MR-FCM for data set sizes varying from 1M to 5M. Since the infrastructure used at TACC does not allow the number of mappers and reducers to be set explicitly, both algorithms are run without specifying the number



(a) 50 mappers and 25 reducers

(b) 100 mappers and 50 reducers

(c) 150 mappers and 75 reducers

(d) 200 mappers and 100 reducers

(e) 250 mappers and 125 reducers

(f) 300 mappers and 150 reducers

Fig. 4 Time results of MR2 with varying data set sizes using different number of mappers and reducers

of mappers/reducers in order to allow for a fair comparison. In addition, a maximum of 10 iterations are used to evaluate the execution time of both algorithms. Table 4 lists the results. We can see that KMeans performs best as expected since it is the computationally less expensive algorithm. As for FKM and MR-FCM being very similar algorithms, we can see that the Mahout FKM implementation scales better than our MR-FCM algorithm. A possible reason

Table 4 Time in seconds comparing KMeans, FKM, and MR-FCM for varying data set sizes

Data set	Time (KMeans)	Time (FKM)	Time (MR-FCM)
1M	352	769	797
2M	365	773	925
3M	368	789	1103
4M	379	846	1318
5M	388	875	1683

for this might be that the Mahout library uses the vector format of the data set whereas the MR-FCM does not.

6 Conclusions

Since current clustering algorithms can not handle big data, there is a need for scalable solutions. Fuzzy clustering algorithms have shown to outperform hard clustering algorithms. Fuzzy clustering assigns membership degrees between 0 and 1 to the objects to indicate partial membership. This paper investigated the parallelization of the FCM algorithm and outlined how the algorithm can be parallelized using the MapReduce paradigm that was introduced by Google. Two MapReduce jobs are necessary for the parallelization since the calculation of the centroids need to be performed before the membership matrix can be calculated.

The accuracy of the MR-FCM algorithm was measured in terms of purity and compared to different clustering algorithms (both hard clustering and fuzzy clustering techniques) showed to produce comparable results.

The experimentation and scalability analysis revealed that the optimal utilization is achieved for the first MapReduce job using 7 mappers and 7 reducers, which is equal to the number of clusters in the data set. Furthermore, it was shown that for the second MapReduce job the best utilization is achieved when using half the number of mappers for the reducers. A factor of 4 in terms of speedup was achieved. The scalability analysis showed that for the data sets investigated (100K up to 900K), a nearly linear increase can be observed when 250 mappers and 125 reducers and more are used. Another evaluation compared the two Mahout algorithms KMeans and FKM with MR-FCM for data set sizes varying from 1M to 5M. This comparison showed that KMeans, being the less computationally expensive algorithm performed best. FKM and MR-FCM are computationally very similar, however, the Mahout FKM algorithm showed to scale better than the MR-FCM algorithm.

Overall, with the implementation we have shown how a FCM algorithm can be parallelized using the MapReduce framework, and the experimental evaluation demonstrated that comparable purity results can be achieved. Furthermore, the MR-FCM algorithm scales well with increasing data set sizes as shown by the scalability analysis conducted.

Future work includes applying the MR-FCM algorithm to different clustering data sets emphasizing on the purity and scalability. Furthermore, larger data set sizes containing GBs of data should be investigated. For this however, another Hadoop cluster needs to be utilized where big data sets can be processed to achieve larger data clustering.

Acknowledgements The author acknowledges the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper.

References

1. G. Bell, A. J. G. Hey, and A. Szalay, Beyond the Data Deluget, Science 323 (AAAS, 6/3/2009), 1297-8. DOI 10.112/science.1170411.
2. A. Ghosh, and L. C. Jain, Evolutionary Computation in Data Mining Series: Studies in Fuzziness and Soft Computing, vol. 163, Springer, 2005.
3. P. Tan, M. Steinbach, and V. Kumar, Introduction to Data Mining, Addison-Wesley, May 2005. (ISBN:0-321-32136-7)
4. H. Jabeen, and A. R. Baig, Review of Classification Using Genetic Programming, International Journal of Engineering Science and Technology, vol. 2. no. 2. pp.94-103, 2010.
5. J. Han, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN:1558609016.
6. R. Babuska, Fuzzy Clustering Lecture, last retrieved from: `http://homes.di.unimi.it/~valenti/SlideCorsi/Bioinformatica05/Fuzzy-Clustering-lecture-Babuska.pdf` on October 2014.
7. J. C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, Kluwer Academic Publishers Norwell, MA, USA, 1981.
8. A. K. Jain, and R. C. Dubes, Algorithms for Clustering Data, Prentice-Hall, Inc., Upper Saddle River, NJ, 1988.
9. E. H. Ruspini, Numerical methods for fuzzy clustering, Information Sciences. 2, 319350, 1970.
10. M. S. Yang, A survey of fuzzy clustering, Math. Comput. Modelling, 18, pp. 1-16, 1993.
11. H. S. Lee, Automatic clustering of business process in business systems planning, European Journal of Operational Research, 114, pp. 354-362, 1999.
12. G. J. Klir, B. Yuan, Fuzzy Sets and Fuzzy Logic Theory and Application, Prentice Hall PTR, Upper Saddle River, NJ, 1995.
13. A. Rosenfeld, Fuzzy graphs, in: L. A. Zadeh, K. S. Fu, M. Shimura (Eds.), Fuzzy Sets and their Applications to Cognitive and Decision Processes, Academic Press, New York, 1975.
14. D. W. Matula, Cluster analysis via graph theoretic techniques Proceedings of the Louisiana Conference on Combinatorics, Graph Theory and Computing, Winnipeg, 1970.

15. J. C. Dunn, A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters, Journal of Cybernetics 3: 32-57, 1973.
16. V. P. Guerrero-Bote, C. Lopez-Pujalte, F. de Moya-Anegon, V. Herrero-Solana, Comparison of neural models for document clustering, Int. Journal of Approximate Reasoning, vol. 34, pp. 287-305, 2003.
17. I. Gath, and A. B. Geva, Unsupervised optimal fuzzy clustering, IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(7), pp. 773-781, 1989.
18. J. C. Bezdek, C. Coray, R. Gunderson and J. Watson, Detection and Characterization of Cluster Substructure - Linear Structure, Fuzzy c-Varieties and Convex Combinations Thereof, SIAM J. Appl. Math., vol. 40, no. 2, pp. 358-372, 1981.
19. Y. Yang, and S. Huang, Image Segmentation by Fuzzy C-Means Clustering Algorithm with a Novel Penalty Term, in Computing and Informatics, vol. 26, pp. 17-31, 2007.
20. W. Cai, S. Chen, and D. Zhang, Fast and robust fuzzy c-means clustering algorithms incorporating local information for image segmentation, Pattern Recognition, vol. 40, no. 3, pp. 825-838, 2007.
21. T. H. Sarma, P. Viswanath, B. E. Reddy, A hybrid approach to speed-up the k-means clustering method, Int. J. Mach. Learn. & Cyber. 4:107-117, 2013.
22. J. Dean, and S. Ghemawat, Mapreduce: simplified data processing on large clusters, in *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI'04, pp. 10–10, 2004.
23. Y. He, H. Tan, W. Luo, S. Feng, and J. Fan, Mr-dbscan: a scalable mapreduce-based dbscan algorithm for heavily skewed data, *Frontiers of Computer Science*, vol. 8, no. 1, pp. 83–99, 2014.
24. W. Zhao, H. Ma, Q. He, Parallel k-means clustering based on mapreduce, in *Proceedings of the CloudCom'09*, Berlin, Heidelberg: Springer-Verlag, pp. 674–679, 2009.
25. P. Zhou, J. Lei, and W. Ye, Large-scale data sets clustering based on mapreduce and hadoop, *Computational Information Systems*, vol. 7, no. 16, pp. 5956–5963, 2011.
26. H.-G. Li, G.-Q. Wu, X.-G. Hu, J. Zhang, L. Li, X. Wu, K-means clustering with bagging and mapreduce, in *Proceedings of the 2011 44th Hawaii International Conference on System Sciences*, Washington, DC, USA: IEEE Computer Society, pp. 1–8, 2011.
27. S. Nair, and J. Mehta, Clustering with apache hadoop, in *Proceedings of the International Conference, Workshop on Emerging Trends in Technology*, ICWET'11, New York, NY, USA: ACM, pp. 505–509, 2011.
28. S. Papadimitriou, and J. Sun, Disco: Distributed co-clustering with mapreduce: A case study towards petabyte-scale end-to-end mining, in *Proc. of the IEEE ICDM'08*, Washington, DC, USA, pp. 512–521, 2008.
29. A. Ene, S. Im, B. Moseley, Fast clustering using mapreduce, in *Proceedings of KDD'11*, NY, USA: ACM, pp. 681–689, 2011.

30. J. Yang, and X. Li, Mapreduce based method for big data semantic clustering, in *Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics*, SMC'13, Washington, DC, USA: IEEE Computer Society, pp. 2814–2819, 2013.
31. F. Cordeiro, C. Traina Jr, A. J. M. Traina, J. Lopez, U. Kang, C. Taloutsos, Clustering very large multi-dimensional datasets with mapreduce, in *Proceedings of KDD'11*, NY, USA: ACM, pp. 690–698, 2011.
32. Simone A. Ludwig, Clonal selection based fuzzy C-means algorithm for clustering, GECCO '14 Proceedings of the 2014 conference on Genetic and evolutionary computation, Pages 105-112.
33. MPI (Message Passing Interface). `http://www-unix.mcs.anl.gov/mpi/`
34. PVM (Parallel Virtual Machine). `http://www.csm.ornl.gov/pvm/`
35. M. V. Modenesi, M. C. A. Costa, A. G. Evsukoff, N. F. Ebecken, Parallel Fuzzy c-Means Cluster Analysis, in Lecture Notes in Computer Science on High Performance Computing for Computational Science - VECPAR 2006, Springer, 2007.
36. J. A. Blackard, Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types, Ph.D. dissertation, Department of Forest Sciences, Colorado State University, Fort Collins, Colorado, 1998.
37. Apache Hadoop. `http://hadoop.apache.org/`.
38. J. Han, Data Mining: Concepts and Techniques, Morgan Kaufmann, San Francisco, CA, USA, 2005.
39. G. Karypis, CLUTO: A clustering toolkit, University of Minnesota, Computer Science, Tech. Rep. 02-017, 2003.
40. T. C. Havens, R. Chitta, A. K. Jain, J. Rong, Speedup of fuzzy and possibilistic kernel c-means for large-scale clustering, 2011 IEEE International Conference on Fuzzy Systems (FUZZ), pp. 463-470, June 2011.
41. R. Hathaway and J. Bezdek, Optimization of clustering criteria by reformulation, IEEE Trans. Fuzzy Systems, vol. 3, pp. 241245, 1995.
42. Mahout library, `http://mahout.apache.org/`.