Parallel Fuzzy c-Means Cluster Analysis

Marta V. Modenesi; Myrian C. A. Costa, Alexandre G. Evsukoff and Nelson F. F. Ebecken

COPPE/Federal University of Rio de Janeiro,
P.O.Box 68506, 21945-970 Rio de Janeiro RJ, Brazil
Tel: (+55) 21 25627388, Fax: (+55) 21 25627392
modenesi@lamce.ufrj.br, myrian@nacad.ufrj.br,
evsukoff@coc.ufrj.br, nelson@ntt.ufrj.br

Abstract. This work presents an implementation of a parallel Fuzzy c-means cluster analysis tool, which implements both aspects of cluster investigation: the calculation of clusters' centers with the degrees of membership of records to clusters, and the determination of the optimal number of clusters for a given dataset using the PBM index.

Topics of Interest: Unsupervised Classification, Fuzzy c-Means, Cluster and Grid Computing.

1. Introduction

The huge amount of data generated by data intensive industries such as Telecommunications, Insurance, Oil & Gas exploration, among others, has pushed Data Mining algorithms through parallel implementations [1, 2]. One requirement of data mining is efficiency and scalability of mining algorithms. Therefore, parallelism can be used to process long running tasks in a timely manner.

There are several different parallel data mining implementations being used or experimented, both in distributed and shared memory hardware [3], as so as in grid environments [4]. All the main data mining algorithms have been investigated, such as decision tree induction [5], fuzzy rule-based classifiers [6, 7], neural networks [8, 9], association rules' mining [10, 11] and clustering [12, 13].

Data clustering is being used in several data intensive applications, including image classification, document retrieval and customer segmentation (among others). Clustering algorithms generally follows hierarchical or partitional approaches [14]. For the partitional approach the k-means and its variants, such as the fuzzy c-means algorithm [13], are the most popular algorithms.

Partitional clustering algorithms require a large number of computations of distance or similarity measures among data records and clusters centers, which can be very time consuming for very large data bases. Moreover, partitional clustering algorithms generally require the number of clusters as an input parameter. However, the number of clusters usually is not known *a priori*, so that the algorithm must be

executed many times, each for a different number of clusters and uses a validation index to define the optimal number of clusters. The determination of the clusters' numbers and centers present on the data is generally referred to as cluster analysis.

Many cluster validity criteria have been proposed in the literature in the last years [16, 17 and 18]. Validity indexes aim to answer two important questions in cluster analysis: (*i*) how many clusters are actually present in the data and (*ii*) how good the partition is. The main idea, present in most of the validity indexes, is based on the geometric structure of the partition, so that samples within the same cluster should be compact and different clusters should be separate. When the cluster analysis assigns fuzzy membership functions to the clusters, "fuzziness" must be taken in account in a way that the less fuzzy the partition is the better.

Usually, parallel the implementations of clustering algorithms [12, 13] only consider strategies to distribute the iterative process to find the clusters' centers. In this work, the entire cluster analysis is investigated, including the determination of the clusters' centers and the optimal number of clusters.

The paper is organized as follows: next section the fuzzy c-means algorithm is sketched. The cluster validation index, known as the PBM index, is presented in section three. The parallel implementation of the cluster analysis is presented in section four. The results obtained with this approach considering scalability and speed-up are presented in section five. Final conclusions and future works are discussed in section six.

2. The Fuzzy c-Means Algorithm

The Fuzzy c-means (FCM) algorithm proposed by Bezdek [15] is the well known fuzzy version of the classical ISODATA clustering algorithm.

Consider the data set $T = \{(\mathbf{x}(t)),\ t = 1..N\}$, where each sample contains the variable vector $\mathbf{x}(t) \in R^p$. The algorithm aims to find a fuzzy partition of the domain into a set of K clusters $\{C_1 \ldots C_K\}$, where each cluster C_i is represented by its center's coordinates' vector $\mathbf{w}_i \in R^p$.

In the fuzzy cluster analysis, each sample in the training set can be assigned to more than one cluster, according to a value $u_i(t) = \mu_{C_i}(\mathbf{x}(t))$, that defines the membership of the sample $\mathbf{x}(t)$ to the cluster C_i.

The FCM algorithm computes the centers' coordinates by minimizing the objective function J defined as:

$$J(m, \mathbf{W}) = \sum_{t=1..N} \sum_{i=1..K} u_i(t)^m \, d(\mathbf{x}(t), \mathbf{w}_i)^2 \tag{1}$$

where $m > 1$. The *m* parameter, generally referred as the "fuzziness parameter", is a parameter to adjust the effect of membership values and $d(\mathbf{x}(t), \mathbf{w}_i)$ is a distance

measure, generally the Euclidean distance, from the sample $\mathbf{x}(t)$ to the cluster's center $\mathbf{w}_i$.

The membership of all samples to all clusters defines a *partition matrix* as:

$$U = \begin{bmatrix} u_1(1) & \cdots & u_K(1) \\ \vdots & \ddots & \vdots \\ u_1(N) & \cdots & u_K(N) \end{bmatrix}. \tag{2}$$

The partition matrix is computed by the algorithm so that:

$$\forall \mathbf{x}(t) \in T, \sum_{i=1..K} u_i(t) = 1.. \tag{3}$$

The FCM algorithm computes interactively the clusters centers coordinates from a previous estimate of the partition matrix as:

$$\mathbf{w}_i = \frac{\sum_{t=1..N} u_i(t)^m . \mathbf{x}(t)}{\sum_{t=1..N} u_i(t)^m}. \tag{4}$$

The partition matrix is updated as:

$$u_i(t) = \frac{1}{\sum_{j=1..K} \left(\frac{d(\mathbf{x}(t), \mathbf{w}_i)}{d(\mathbf{x}(t), \mathbf{w}_j)} \right)^{\frac{2}{(m-1)}}}. \tag{5}$$

The FCM algorithm is described as follows:

0. Set $m > 1$, $K \geq 2$ and initialize the cluster centers' coordinates randomly, initialize the partition matrix as (5).
1. For all clusters $(2 \leq i \leq K)$, update clusters' centers coordinates as (4).
2. For all samples $(1 \leq t \leq N)$ and all clusters $(2 \leq i \leq K)$, update the partition matrix as (5).
3. Stop when the norm of the overall difference in the partition matrix between the current and the previous iteration is smaller than a given threshold ε; otherwise go to step 1.

In fuzzy cluster analysis the FCM algorithm computes clusters centers' coordinates and the partition matrix from the specification of the number of clusters K that must be given in advance. In practice, the FCM algorithm is executed to various values of K, and the results are evaluated by a cluster validity function, as described next.

3. Cluster Validity Index

In this work, the PBM index [18] is used to evaluate the number of clusters in the data set. The PBM index is defined as a product of three factors, of which the maximization ensures that the partition has a small number of compact clusters with large separation between at least two of them. Mathematically the PBM index is defined as follows:

$$PBM(K) = \left(\frac{1}{K} \cdot \frac{E_1}{E_K} . D_K \right)^2 \tag{6}$$

where K is the number of clusters.

The factor E_1 is the sum of the distances of each sample to the geometric center of all samples $\mathbf{w}_0$. This factor does not depend on the number of clusters and is computed as:

$$E_1 = \sum_{t=1..N} d(\mathbf{x}(t), \mathbf{w}_0). \tag{7}$$

The factor E_K is the sum of within cluster distances of K clusters, weighted by the corresponding membership value:

$$E_K = \sum_{t=1..N} \sum_{i=1..K} u_i(t) d(\mathbf{x}(t), \mathbf{w}_i)^2 \tag{8}$$

and D_K that represents the maximum separation of each pair of clusters:

$$D_K = \max_{i,j=1..K} \left(d(\mathbf{w}_i, \mathbf{w}_j) \right) . \tag{9}$$

The greatest PBM index means the best clustering fuzzy partition. As other indexes, the PBM index is an optimizing index, so that it can be used to search the best number of clusters within a range of number of clusters. The PBM procedure can be described as follows:

0. Select the maximum number of clusters M ;
1. Compute the *PBM* factor E_1 (7)
2. For $K = 2$ to $K = M$, do:
   2.1. Run the FCM algorithm;
   2.2. Compute the *PBM* factors E_K (8) and D_K (9);
   2.3. Compute the $PBM(K)$ index (6)
3. Select the best number of clusters K^* as:

$$K^* = \arg\max\left(PBM(K)\right) \tag{10}$$

The PBM index has achieved a good performance in several data when compared with the Xie-Beni index [16]. This index is thus used as a validity index of the methodology presented in this work.

4. Parallel Cluster Analysis Implementation

The aim of the FCM cluster analysis algorithm is to determine the best partition for the data being analyzed, by investigating different partitions, represented by the partitions' centers. Hence, the cluster analysis must integrate the FCM algorithm and the PBM procedure as described above.

The cluster analysis is an iterative process where the FCM algorithm is computed for a range of number of clusters and the PBM index is computed for every partition generated by the FCM algorithm. When all partitions have been computed, the partition corresponding to the maximum PBM index is chosen as the best partition for the data.

The most complex computation in the FCM algorithm is the distance computation from each sample $\mathbf{x}(t)$ to all clusters' centers $\mathbf{w}_i$, $i = 1..K$. This computation is performed every interaction, for all records in the dataset. Aggregates of the distances are used to compute the new centers' estimate (4), the fuzzy partition's matrix (5) and the PBM factors E_1 (7) and E_K (8). These are the steps of the FCM cluster analysis that should be parallelized.

The parallel FCM cluster analysis procedure is sketched in Fig 1 and described by the following sequence:

Step 1. (Master processor): Splits the data set equally among the available processors so that each one receives N/p records, where N is the number of records and p is the number of processes

Step 2. (All processors): Compute the geometrical center of its local data and communicate this center to all processors, so that every processor can compute the geometrical center of the entire database. Compute the PBM factor E_1 (7) on local data and send it to root.

Step 3. (Master processor): Sets initial centers and broadcasts them, so that all processors have the same clusters' centers values at the beginning of the FCM looping.

Step 4. (All processors): Until convergence is achieved compute the distances from each record in the local dataset to all clusters' centers; update the partition matrix as (5), calculate new clusters' centers as (4).

Step 5. (All processors): Compute the PBM factor E_K (8) on its local data and send it to root.

Step 6. (Master Processor): Integrates the PBM index as (6) and stores it. If the range of number of clusters is covered, stops, otherwise returns to *Step3*.

The procedure described above is computed for each number of clusters in the cluster analysis, so that the procedure is repeated as many times as the desired range of numbers of clusters, so that the PBM index, as a function of the number of centers, is computed. The best partition is the one corresponding to the largest value of the PBM index.

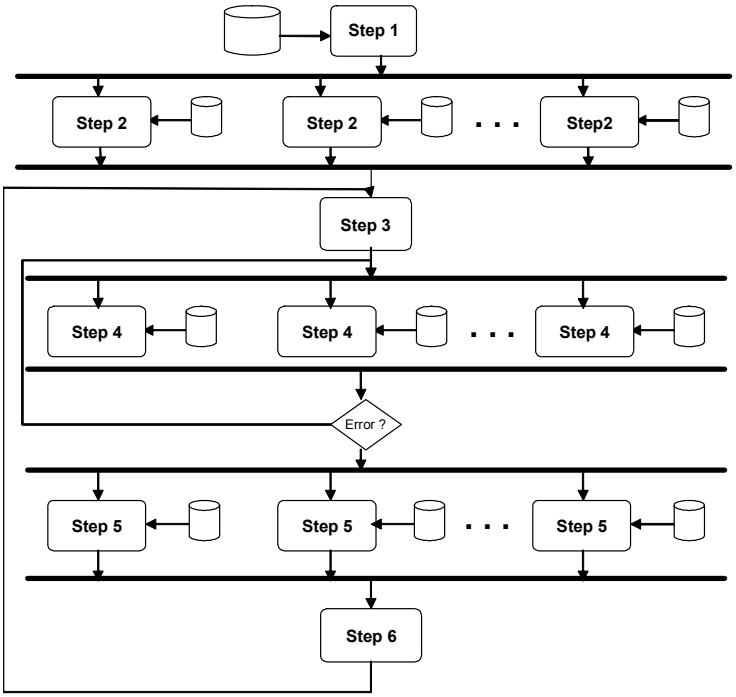

Fig 1. The parallel FCM cluster analysis

5. Results and Discussion

5.1. Environment

Two machines were used for execution and performance analysis of this work: the PC Cluster Mercury and the SGI Altix 350, both from the High Performance Computing Center (NACAD) of COPPE/UFRJ. The PC cluster has 16 dual Pentium III, 1 GHz, processor nodes interconnected via Fast Ethernet and Gigabit networks and 8GB of total memory. The SGI Altix 350 has 14 Intel Itanium2 cpus with 28 Gbytes of RAM (shared - NUMA) and 360 Gbytes of disk storage.

In both machines the execution is controlled by PBS (Portable Batch System) job scheduler avoiding nodes sharing during execution and Linux Red Hat runs on processing and administrative nodes. The application was developed using C programming language and Message Passing Interface (MPI).

5.2. The Cluster Mercury Results and Speed-up Analysis

The speed up evaluation of the FCM cluster analysis algorithm was made in two test steps. In the first one, the objective was to observe the algorithm behavior increasing the number of records. In the second test the objective was to observe the behavior of the algorithm when increasing the number of variables and of the range of partitions.

Test 1. Datasets of different line sizes were used. The datasets had 1.000, 12.500, 50.000, 65.000 and 100.000 records (lines) and size of 38Kb, 500kb, 1.88MB, 2.5MB and 3.76MB. A fixed number of variables and a fixed range of clusters were used. The datasets had 18 variables (columns). The evaluation was made performed considering 9 partitions calculated from $K = 2$ to $K = 10$ clusters' centers, used for the PBM index calculation. The speed-up results are shown in Table 1.

Table 1 - Speed-up results for Cluster Mercury

Datasets	Number of Processors							
	1	2	3	4	5	6	7	8
38Kb	1	1.84	1.97	2.24	2.01	1.94	1.95	1.91
500Kb	1	1.96	2.80	3.67	4.33	4.97	5.67	6.12
1.88MB	1	1.96	2.89	3.80	4.62	5.48	6.32	7.11
2.50MB	1	1.96	2.90	3.82	4.68	5.55	6.41	7.21
3.76MB	1	1.96	2.91	3.83	4.72	5.60	6.47	7.30

The algorithm's speed up when processing the smaller dataset was clearly worse than the others. Its highest speed up value was 2.24 when using 4 processors. The algorithm showed higher speed up values for a larger number of processors when processing datasets with larger number of records. When processing a small number of records, communications are too costly compared to the advantage of parallelizing the calculations of distances to clusters, and the benefits of parallelism do not happen.

As showed in Table 1, using 8 processors, speed up values of more than 7.0 were achieved for databases with more than 1.88MB, which is a very good gain for the overall parallel process. The speed up of the parallel program against the correspondent sequential one has an efficiency factor that gets as much closer to 1 as the database records increase, when considering a growing number of processors.

Test 2. To investigate the effect of the number of variables in the parallel process, two datasets were used: one of 50.000 lines and 10 variables of 1.07MB and the other with 50.000 lines and 40 variables (columns) of 4.12MB. The two datasets were processed using different ranges of clusters. The first computation of the algorithm was made with one partition of 2 clusters. Ranges of 2 to 4 clusters, of 2 to 8 clusters, 2 to 16 clusters and of 2 to 32 clusters had been used in each other algorithm computation. All processing were executed for 1, 2, 4, 6, and 8 processors. Results are presented in Table 2 and Table 3.

The second test showed that the number of variables have also an impact on the overall performance of the algorithm. The processing of datasets with small number of variables gets smaller speed up values. The processing of datasets with larger number of variables results in better speed up values when using a larger number of processors.

The same happens when refereeing to the range of clusters to form the partitions. The bigger is the range of number of clusters' centers that have to be investigated, the greater is the number of partitions that will have to be calculated. Also, as larger are the clusters' numbers, more computation is involved in distance calculations from records to clusters. As the calculations increase in the processing, the parallel algorithm benefits show up clearly. To use a higher number of processors is as much interesting, in time savings and processing acceleration, as the range of number of clusters increases.

Table 2 – Speed up for dataset of 50.000 lines x 10 variables

Clusters' Ranges	Number of Processors				
	1	2	4	6	8
2 clusters	1	1.79	2.82	3.44	3.79
from 2 to 4	1	1.91	3.47	4.78	5.83
from 2 to 8	1	1.94	3.71	5.26	6.66
from 2 to 16	1	1.95	3.78	5.46	7.02
from 2 to 32	1	1.96	3.81	5.51	7.19

Table 3 – Speed up for dataset of 50.000 lines x 40 variables

Clusters' Ranges	Number of Processors				
	1	2	4	6	8
2 clusters	1	1.78	2.80	3.45	3.93
from 2 to 4	1	1.92	3.55	4.94	6.22
from 2 to 8	1	1.96	3.81	5.54	7.20
from 2 to 16	1	1.98	3.89	5.73	7.53
from 2 to 32	1	1.98	3.92	5.81	7.67

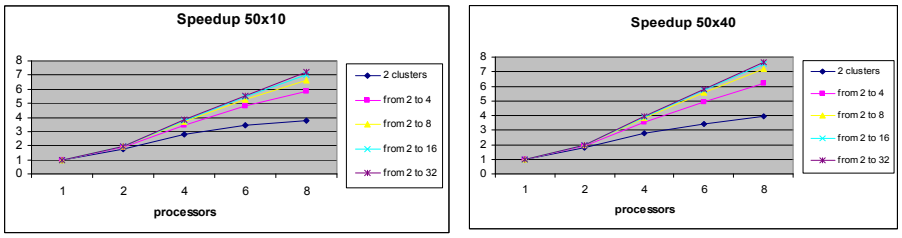

Fig. 2 – Speedup graphs for different files sizes.

The tests of the parallel Fuzzy c-Means cluster analysis tool on the Mercury Cluster hardware has shown that it is scalable for parallel cluster analysis processing on these machines for databases of bigger sizes.

5.3. The Altix Machine Tests

The Tests Description

There were used twelve different files to proceed with the programs test in the Altix machine. The files' dimensions schemas are presented in Table 4 and the files sizes are presented in Table 5.

Table 4 – Files dimensions

Registers	Variables			
50.000	50	100	150	200
100.000	50	100	150	200
200.000	50	100	150	200

Table 5 – Files sizes

File Id	Records (in thousands) x Variables	Size(in MB)
1	50 x 50	5.138
2	50 x 100	10.227
3	50 x 150	15.317
4	50 x 200	20.406
5	100 x 50	10.347
6	100 x 100	20.552
7	100 x 150	30.730
8	100 x 200	40.909
9	200 x 50	20.747
10	200 x 100	41.104
11	200 x 150	61.469
12	200 x 200	81.817

There were made 288 tests using this files base to test the Parallel Fuzzy c-Means Cluster Analysis program behavior with greater data volumes. Each file was tested with clusters' ranges of 2, 4, 8 and 16, and each range of clusters was tested with processors varying from 1 to 6.

Parallel Processing Time Analysis

One of this work's goals is to create a useful tool for data analysis in the oil and gas field, where huge volumes of data need to be investigated in order to find out oil reservoir information. So, it is necessary to investigate the programs behavior when number of records and variables increase.

It was observed that when the number of records in the database grows, the processing time grows as well, proportionally to the increase of records (Fig.3).

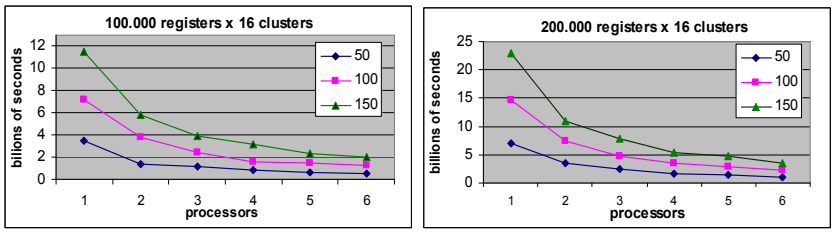

Fig. 3 - Increasing of time when increasing number of variables.

The same behavior occurred when increasing the number of variables: processing time grows at the same rate of the increasing of the variables as can be seen in Fig.4. Nevertheless, the parallel Fuzzy c-Means Cluster Analysis program has the same behavior for all problem sizes being investigated. The parallel approach decreases the time processing for all files' sizes, but the time savings is more meaningful for larges databases because it is proportional to the problem size.

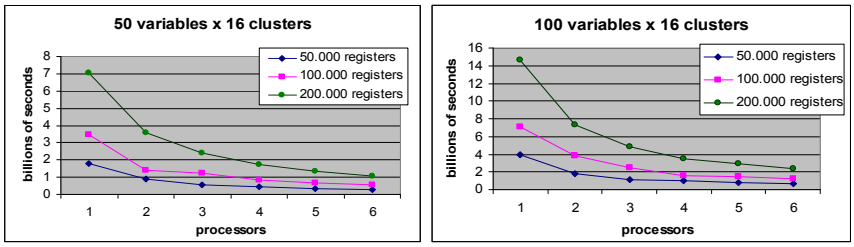

Fig. 4 - Increasing of time when increasing number of records.

Speed Up and Efficiency Analysis

In the Altix machine tests the processing for the smaller files size for only 2 clusters did not presented a good speedup curve. This can be understood considering that the computational effort for processing only one partition of two clusters is significantly smaller than the communications' cost involved in the overall process. Measurements show it clearly as can be visualized in Figure 5 bellow.

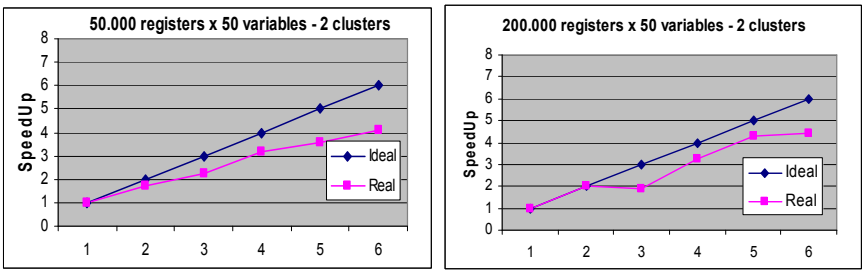

Fig. 5 – Speedup of processing one partition of two clusters using files of 50.000 registers and 50 variables and 200.000 registers and 50 variables.

Processing with bigger files improved the speedup curve because the computational time tends to be greater than communications time involved in the process for bigger clusters interval. Tests in the Altix machine presented in a few cases a super linear speed-up value as an example showed in Figure 6.

This behavior could be explained because the distribution of one bigger file through several processors produces smaller files for each processor. The architecture of the Altix machine provides a performance improvement of an application with this feature. If the file is so small that can be stored in the memory cache, the computational time decreases and the speed-up becomes super linear.

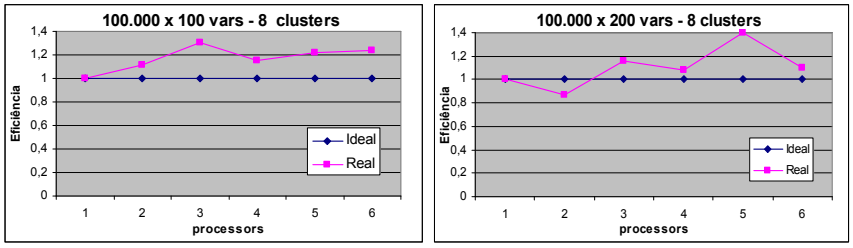

Fig. 6 – Super linear efficiency values.

This result shows that the Parallel Fuzzy c-Means Cluster Analysis tool scales well and can be used successfully processing bigger datasets.

Files Size versus Parallelization Benefits

In order to have an indication of whether it is efficient to use parallel processing or not, all the 288 tests results in Altix hardware were used as records in a dataset for a decision tree induction classification algorithm.

The input variables for the decision tree induction algorithm were the number of records, the number of variables and the number of clusters. The class output was computed by the efficiency of the parallel processing, defined as the ratio between the speed-up and the number of processors. For each test record, if the efficiency was greater than 0.8 the test was set to YES, meaning that the parallel processing is efficient, otherwise the class was set to NO.

For the analysis, the decision tree induction algorithm J48, was used within the Weka open source data mining workbench [19]. The J48 algorithm is a Java implementation of the classical C4.5 algorithm, developed by Quinlan [20].

The resulting decision tree is shown in Figure 7, where it is clearly shown that the parallel processing is not efficient for 2 clusters and is efficient for more than 8 clusters. In the case of 4 clusters, the parallel processing could be efficient or not depending on the number of records and/or the number of variables.

Although it is not a definitive answer for the load optimization of the cluster analysis, the decision tree shows that the number of clusters is the parameter that affects mostly the efficiency of the parallel processing. As a current cluster analysis must compute the FCM algorithm for a range of number of clusters, in order to determine the optimal number of clusters, it is preferable to not parallelize the processing for small number of clusters and use more processors as the number of clusters increases.

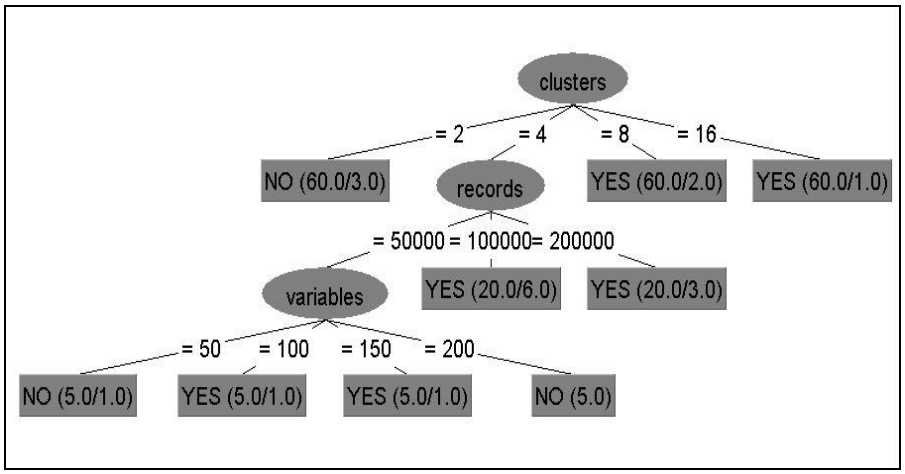

Fig 7 – Weka's J48 decision tree.

6. Conclusions

This work has presented a parallel implementation of FCM cluster analysis where both the determination of clusters' centers and the number of clusters are optimized by the algorithm. The main contribution of this work is the integration of the cluster validation index in the optimization process, allowing the optimization of the overall parallel process.

The implementation and performance tests were made in two different hardware architectures: the first on low cost distributed memory hardware, a PC Cluster, and the second on a machine of bigger computational power, the Altix 350.

The parallel Fuzzy c-Means Cluster Analysis tool behaviors in a scalable manner presenting good speedup and efficiency values in both hardware, showing that it can be used as a valuable and efficient tool for improve processing time in knowledge discovery in very bigger databases.

Acknowledgements

This work has been supported by the Brazilian Research Council (CNPq), by the Brazilian Innovation Agency (FINEP) and by the National Petroleum Agency (ANP). The authors are grateful to High Performance Computing Center (NACAD-COPPE/UFRJ) where the experiments were performed.

References

1. M. S. R. Sousa, M. Mattoso and N. F.F. Ebecken (1999). Mining a large database with a parallel database server. *Intelligent Data Analysis* 3, pp. 437-451.
2. M. Coppola, and M. Vanneschi. (2002). High-performance data mining with skeleton-based structured parallel programming. Parallel Computing 28, pp. 783-813.
3. R. Jin, G. Yang, and G. Agrawal (2005). Shared Memory Parallelization of Data Mining Algorithms: Techniques, Programming Interface, and Performance. *IEEE Transaction on Knowledge and Data Engineering*, vol. 17, no. 1, pp. 71-89.
4. M. Cannataro, A. Congiusta, A. Pugliese, D. Talia, P. Trunfio (2004). Distributed data mining on grids: services, tools, and applications. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 34, no. 6, pp. 2451 – 2465.
5. K. Kubota, A. Nakase, H. Sakai and S. Oyanagi (2000). Parallelization of decision tree algorithm and its performance evaluation. *Proceedings of the Fourth International Conference on High Performance Computing in the Asia-Pacific Region*, vol. 2, pp. 574 – 579.
6 M. W. Kim, J. G. Lee and C. Min (1999). Efficient fuzzy rule generation based on fuzzy decision tree for data mining. *Proceedings of the IEEE International Fuzzy Systems Conference FUZZ-IEEE '99*. pp1223 – 1228.
7 A. Evsukoff, M. C. A. Costa and N. F. F. Ebecken (2004). Parallel Implementation of Fuzzy Rule Based Classifier. *Proceedings of the VECPAR'2004*, vol. 2, pp. 443-452.
8 P. K. H. Phua and D. Ming. (2003). Parallel nonlinear optimization techniques for training neural networks. *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1460 - 1468.
9 M. C. A. Costa and N. F. F. Ebecken (2001). A Neural Network Implementation for Data Mining High Performance Computing. *Proceedings of the V Brazilian Conference on Neural Networks*, pp. 139-142.
10 R. Agrawal and J. C. Shafer (1996).Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 962 - 969.
11 L. Shen, H. Shen and L. Cheng (1999). New algorithms for effcient mining of association rules. *Information Sciences* 118, pp. 251 – 268.
12 B. Boutsinas and T. Gnardellis (2002). On distributing the clustering process. *Pattern Recognition Letters* 23, pp. 999–1008.
13 S. Rahimi, M. Zargham, A. Thakre and D. Chhillar (2004) A parallel Fuzzy C-Mean algorithm for image segmentation. *Proceedings of the IEEE Annual Meeting of the Fuzzy Information NAFIPS '04*, vol. 1, pp. 234 – 237.
14 A. K. Jain, M. N. Murty and P. J. Flynn (1999). Data clustering: a review. *ACM Computing Surveys*, vol. 31, no. 3. pp. 264-323.
15 J. C. Bezdek (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York, Plenum.
16 X. L. Xie and G. A. Beni (1991). Validity measure for fuzzy clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 3 no. 8, pp. 841–846.

17 J. Bezdek and N.R. Pal (1998). Some new indexes of cluster validity. *IEEE Trans. Systems Man and Cybernetics B*, vol. 28, pp. 301–315.
18 M. K. Pakhira, S. Bandyopadhyay and U. Maulik (2004). Validity index for crisp and fuzzy clusters. *Pattern Recognition*, vol. 37, pp. 487-501.
19 I. H. Witten and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edition, Morgan Kaufmann, San Francisco.
20 R. Quinlan (1993). *C4.5 – Programs for Machine Learning*. Morgan Kaufmann, San Francisco.