Distributed Unsupervised Learning Using the MULTISOFT Machine

Giuseppe Patané [a] and Marco Russo [b,1]

[a] *Institute of Computer Science and Telecommunications, Faculty of Engineering, University of Catania, Viale A. Doria 6, 95125 Catania, Italy and INFN, Section of Catania, Corso Italia 57, 95129 Catania, Italy; e-mail: gpatane@ai.unime.it*

[b] *Department of Physics, University of Messina, Contrada Papardo, Salita Sperone 31, 98166 Messina, Italy and INFN, Section of Catania, Corso Italia 57, 95129 Catania, Italy; e-mail: marco.russo@ct.infn.it*

---

[1] Marco Russo, Department of Physics, University of Messina, Contrada Papardo, Salita Sperone 31, 98166 Messina, Italy; e-mail: marco.russo@ct.infn.it

Distributed Unsupervised Learning Using the MULTISOFT Machine

Abstract

Unsupervised learning using K-means techniques is successfully employed in several application fields. When the training set and the number of reference vectors increases, the computational effort can become prohibitive for monoprocessor computers. This paper illustrates the parallelization of two clustering techniques using the MULTISOFT machine, a commodity supercomputer, built at the University of Messina. The particular management policy of the MULTISOFT machine and the implementation techniques have shown very interesting results: the speedup increases together with the complexity of the problem to be solved.

Key words: Unsupervised Learning, Vector Quantization, Clustering, Parallel, Multicomputers.

1 Introduction

Clustering is an important family of algorithms widely employed in several scientific disciplines and used for a lot of successful applications. Among them there are radial-basis function networks [1], pattern recognition [2], computer vision [3] and classification tasks [4]. Several papers [5,6] demonstrate that, in many cases, clustering is equivalent to Vector Quantization (VQ), a technique often employed in telecommunications and signal compression [7,8]. For this reason, in the remainder of the paper, we will use also the term *Unsupervised Learning* (UL) to indicate one or both of the two approaches. Besides, we will use a symbology and terminology typical of VQ.

Many algorithms are based on the K-means clustering technique [9–11,4,12]. This is a very useful family of algorithms (both fuzzy and hard) and, generally speaking, does not require many hardware resources. However, sometimes, we must deal with very large codebooks for very large training sets. In these cases, it could be necessary to work with very powerful computers, even with multiprocessor-based ones. But we know that these kinds of machines are, generally, very expensive. One intermediate choice is to use multicomputers. In particular, a solution that is spreading very fast consists of commodity supercomputers.

This work deals with the parallelization of the standard K-means algorithm (also known as the LBG algorithm [10]) and its enhancement (the ELBG algorithm [13–16]) on the MULTISOFT machine. This is the name of a cheap, but powerful, farm of Personal Computers (PCs) implemented at the University of Messina.

In literature, we can find some parallel algorithms for clustering, although they have been treated in a different way. For example, Ratha et al. [17] proposed a genetic algorithm for clustering. They used 5 Sun SPARC 10 stations and, for the very simple IRIS [18] problem, obtained a speedup of 4.1. Ancona et al. [19] described a parallel implementation of NNs based on vector quantization. They used a toroidal-mesh architecture composed of 6 transputers of the T800 family, using inter-transputer links at 20 Mbit/s. The reached speedup ranged from 3 to 4.2.

This paper is divided into two different parts. In the first part, the authors go into detail regarding the management of the MULTISOFT machine. The policy adopted permits a very fast increase of the number of the Processing Elements (PEs), that is the number of PCs. It is based on a completely centralized administration of the system, but particular care was taken in order to minimize the network bandwidth required in normal operative conditions.

Several similar machines exist in the world [20] [2] . They are also based on a centralized administration, but a quota of the interconnection network bandwidth is needed for the sharing of the file system. This aspect is very important above all for machines like the MULTISOFT one, where the total bandwidth is very low. Generally, the interconnection network in multicomputers is the bottleneck of all well-implemented distributed algorithms, so its under-utilization directly affects the maximum value of the achievable speedup.

In the second part of the manuscript, the authors describe their implementation of the parallel version of both LBG and ELBG algorithms. The methodology adopted behaves poorly in the case of simple clustering problems. The lack of a true broadcasting function in the communication system used makes everything worse. However, when the number of required computations grows, making the clustering task more complex, the results are encouraging because the speedup significantly increases. This means that commodity supercomputers are very suitable for very complex unsupervised problems.

2 The MULTISOFT machine

2.1 Hardware organization

The MULTISOFT machine (see Fig. 1) is a cluster of PCs with the LINUX OS. It consists of 32 Processing Elements (PEs or hosts) and a server (duplicated for redundancy) acting also as a gateway towards the external LAN. All of the 32 PEs are mono-processor machines, 7 are based on Pentium-II 233MHz (they are called p233) and 25 on Celeron 366MHz (c366). All of them are endowed with 32 MB of RAM, one hard disk (ranging from 3.2 to 4.3 GBytes) and one Fast-Ethernet network card. One server (Server01) is a bi-processor computer with two Pentium-II 300MHz and two network cards and the other (Server02) is a mono-processor Pentium II 350MHz.

The PEs are interconnected by a dedicated Fast Ethernet network and all of them share the same bus. Physical connections have been realized by means of four 12-port hubs distributed between two levels as shown in Fig. 1. However, the electrical topology is such that all of the 32 hosts are placed on the same bus. The highest level hub is connected to a switch, where the servers are directly connected, too.

Inter-process communication is realized with the Parallel Virtual Machine

[2] A 1000-pentium based machine employed for applications regarding genetic programming is described at http://www.genetic-programming.com/machine1000.html

(PVM) software [3] . It allows the user to see several computers as a large and single virtual machine and every process (task, according to the PVM nomenclature) has a unique identifier (task identifier, tid) inside the virtual machine. PVM uses the transport functionality provided by the TCP/(UDP/)IP that are directly implemented inside the kernel of LINUX. Together with PVM, the graphic tool named XPVM is employed; it allows the visualization of the diagram of the activity of the tasks versus time. It is very useful during the debugging and the testing of parallel algorithms. Further, it allows distributed algorithms to be described well, as will be shown later in this paper.

2.2 Management

For the management of a system with a high number of hosts, all sharing the same configuration, the authors thought it was suitable to pursue a centralized policy allowing the administrator to quickly modify the configuration of the whole system without having to operate on the single machine. Besides, when the system expands, we want to be able to quickly set up the new machines.

The first solution considered was the realization of a system similar to the one described in [20]. All of the hosts were disk-less machines and remote-boot and Network File System (NFS) [4] were employed. In this way, each host shared most of its file-system (physically stored on the server) with all of the other hosts. Files that cannot be shared were stored on the server, too, but were kept in distinct directories, one for each host. These are the files containing information related to the identification and the status of the single host, its peripherals and the processes running on it. Therefore, they are different for each of them and, for this reason, they cannot be shared. Such a policy allows a quick update of the system because each modification effected on the shared portion of the file-system of a host, automatically and immediately, regards all of the other hosts. On the other hand, changes regarding the unshared portions are much less frequent. However, when such a centralized solution is adopted, we have to consider two main problems. The first concerns the access to a NFS, much slower than a local file-system. The second arises from the increase in the network traffic, with the consequent significant reduction of the bandwidth available to inter-process communication.

On the basis of these considerations, the authors found two, apparently, opposite requirements: the centralization of the information (in order to simplify

[3] It is available together with XPVM for free at http://www.netlib.org/pvm

[4] In this section, several protocols for the management of the system (NFS, NIS, bootp, dhcp) are cited. The authors suggest to visit http://www.linux.org/docs/index.html for useful theoretical and practical details about this topic.

the management of the system) and the maximization of the performances by saving bandwidth, so that it is completely at the disposal of inter-process communication.

The solution chosen is a hybrid one between centralized and local management. Under normal conditions of working, all of the hosts use a file-system stored on the local disk. Nevertheless, it is a copy of a file-system that is physically stored on the server. It is the same for everyone and it is downloaded by the hosts in an almost completely automatic way. Such an operation of downloading is necessary only when modifications or failures in the hosts occur. So, we need to take care of the configuration of only two machines: the server and a host. In fact, once a generic host has been configured (upgraded), its file-system is transferred to the server and from there it is, automatically, downloaded by all of the other hosts.

Before describing in greater detail the policy adopted for the management of the system, it is important to point out that, according to the terminology employed, each host can operate in one of the three following modalities:

- **Managing.** It is used for the first installation or if a failure occurs.
- **Service.** It is used to update the hosts.
- **Operative.** It is the normal modality of working.

A host enters a certain modality when it boots. When it is necessary to change the modality, its configuration is opportunely modified so that, after the reboot, it enters the desired modality. Each host can operate in any modality with no console (keyboard, monitor, mouse). Only the server has a console.

The three modalities and the transitions from one modality to another, can be represented by means of a Finite State Machine (FSM), as reported in Fig. 2. The signals that make the host change (or remain in) its state are the following:

- start: it is used to begin the first installation of the host. It indicates that a new PE must be physically and logically added to the system, or that an existing PE has to be reconfigured *ex novo* after a failure.
- power failure (pf);
- reboot: it represents the normal reboot of the host;
- sync: it is used to force the host to download from the server the most recent version of the file-system related to the operative modality;
- host failure (hf): it indicates that the host is not working correctly.

Now, the operations effected in each modality and how the transition from one modality to another occurs will be detailed.

- **Operative modality.** This is the modality where the host, normally, op-

erates. It boots and works by using the local operative file-system (that is the same for all of the hosts). Only the information related to the host identification on the network (its name and its IP address) differs for each of them. However, these data are not physically stored on the host, but are all on the server which communicates them to the host when it boots by means of a protocol such as bootp or dhcp. Even if the host works by using a local file-system, the auditing of the user accounts is effected by the server through the NIS protocol. Besides, the home directories of the users are shared by means of the NFS. However, users have also a personal local directory on each host, so that they can choose to use the shared, the local directory or both of them. If the host is in this state and a reboot or a power failure occur, it reboots and remains in the same state. Instead, if the signal of sync is raised, the host opportunely modifies its configuration and reboots in the service modality.

- **Service modality.** This is a transitory modality. It is used by the hosts to download the file-system that will be used in the operative modality from the server. In this state, the kernel mounts a reduced file-system, identical for all of the hosts, previously downloaded from the server, exactly as it happens for the operative file-system. Also in this case, the information identifying that host in the network are assigned by the server during the boot phase. The service file-system contains only the files that are necessary to the host so that it can download the operative file-system from the server and store it on the local disk. If no power failures or other generic failures occur, the host modifies its configuration and reboots in the operative modality. All of these operations are executed automatically. If a power failure occurs, the host reboots in the service modality, while, if a generic failure occurs, it is rebooted in the managing modality by inserting the proper floppy disk.
- **Managing modality.** When a host operates in this modality, it is able to work without using the local disk, as if it was a disk-less machine. All of the files necessary to effect the remote-booting are stored on a bootable floppy disk. After the kernel has been loaded, the host mounts the file-system related to this modality by NFS. At this point, all of the operations relating to the initialization of the local disk (partitioning and formatting) necessary so that this can hold the service and the operative file-system, are executed. Therefore, the service file-system is downloaded from the server and the configuration is modified so that the host can reboot in the service modality. The reboot gets the hosts into the service modality, from where, if no failures occur, it automatically enters the operative modality. If a power failure or a generic failure occur the host reboots in the managing modality.

In the previous subsection the modalities where each host of the system can operate have been described. Now, the procedures related to its updating will be explained. This is necessary, for example, when a program is upgraded or installed *ex novo*. The modifications are made on any host working in the operative modality. Afterwards, its operative file-system is compressed and transferred to the server. Now, it is enough to raise the "sync" signal for all of the hosts and these will reboot in the service modality. Therefore, as previously described, they download the most recent version of the operative file-system from the server and, when they finish, reboot in the operative modality. So, we can easily transfer the changes made on a single machine to all of the system and in a completely automatic way . We must remember that this would be possible if the hosts mount their whole file-system by NFS but, as previously said, this would make the system work more slowly and would consume transmission bandwidth. Instead, the method proposed allows us to automatically update the system without overloading the network when it works in the normal operative modality.

3 Vector Quantization (VQ)

Definitions. A Vector Quantizer is a function $q : X \longrightarrow Y$ that represents a set of feature vectors $\mathbf{x} \in X \subseteq \Re^m$ by a set, $Y = \{\mathbf{y}_1, ..., \mathbf{y}_{N_C}\}$, of N_C reference vectors in $\Re^m$. Y is called *codebook* and its elements *codewords*. The vectors of X are called also *input patterns* or *input vectors*. According to the previous definitions, we can obtain a partition $\mathcal{S}$ of X constituted by the N_C subsets S_i (called cells): $S_i = \{\mathbf{x} \in X : q(\mathbf{x}) = \mathbf{y}_i\} \quad i = 1, \ldots, N_C$.

The Quantization Error (QE) is the value assumed by $d(\mathbf{x}, q(\mathbf{x}))$, where d is a generic distance operator for vectors. The performance of a vector quantizer is given by its mean QE (MQE). Several functions can be adopted as distortion measures [10]. The most widely adopted is the Euclidean distance and it will be used throughout the paper. Besides, only input data sets constituted by a finite number (N_P) of m-dimensional vectors will be considered.

LBG. LBG was proposed in 1980 [10] by Linde, Buzo and Gray and its original name was Generalized Lloyd Algorithm (GLA) because it extended the Lloyd's technique [21] from mono- to m-dimensional cases. The name LBG comes from the initials of its authors. It is an algorithm, that, at every iteration, generates a quantizer whose MQE is less or equal to the previous one. This is obtained by a process of optimization where the best partition (given the codebook) and the best codebook (given the partition) are alternatively obtained. Several

techniques for calculating an initial codebook exist [10] but, in this paper, we are not interested in this phase. Therefore, we assume that a random choice of the initial codewords is enough for starting-up the algorithm. LBG can be summarized as follows;

(1) an initial codebook is given;
(2) given the codebook, the best partition is calculated (Voronoi partition). It is obtained by assigning each pattern to the nearest codeword to it;
(3) termination condition check: the current MQE of the quantizer is compared to the one at the previous iteration. If it decreases less than a prefixed threshold, the algorithm ends; otherwise it continues;
(4) given the partition calculated at point 2, the best codebook is calculated. Its elements are the geometrical barycentres of the cells; it can be demonstrated that this is the optimal choice when the Euclidean distance is adopted as the distortion measure.
(5) return to step 2.

Enhanced LBG (ELBG). In [15,13], the authors tried to explain why LBG suffers the problem of the local minima deriving from a bad initialization of the codebook. There, they illustrated the technique employed to solve such a problem. It consists in the pursuit of the equalization of the total distortions introduced by each cell. The total distortion of a cell is the sum of the QEs of the vectors belonging to it. The operations executed to try to obtain such an equalization are based on a stochastic process for selecting some cells to be split and joined. They are executed inside another step, called ELBG block, which is placed between points 3 and 4 of LBG. For this reason, their algorithm is called ELBG. The interested reader can see [15,13].

4 Parallelization

In this section the implementation of the parallel versions of LBG and ELBG are described.

In [15], the authors saw that the overhead introduced on the traditional LBG by the execution of the ELBG block is quite low. Besides, the profiling executed on the software written to implement LBG allowed them to see that almost all of the calculation time is spent to determine the Voronoi partition (point 2 of the LBG algorithm). More precisely, this time is spent to calculate the distance between each input pattern and all of the codewords, in order to choose the nearest one. The work presented in this paper focused on the parallelization of this operation by trying to share the load among all of the hosts. Now, the authors will describe the parallel implementation LBG they made and, after, the parallel version of ELBG (that is identical to LBG only with the addition

of the ELBG block).

4.1 LBG

As previously said, during the calculation of the Voronoi partition, each input pattern is assigned to the nearest codeword to it. As this is done by comparing each input vector with all of the codewords, it means that the distance (Euclidean, in this case) between two m-dimensional vectors is calculated $N_P \times N_C$ times. If we have N processors at our disposal, we can launch, on each processor, a process (task, according to the PVM nomenclature) that executes $\frac{N_P \times N_C}{N}$ of distance calculations. In practice, each task keeps in the memory a different portion of the input patterns and the whole codebook. It has to find the nearest codeword only for the portion of input patterns that were assigned to it.

Given N equal hosts, a master-slave policy has been adopted. At each iteration, the master calculates, together with the slaves, the Voronoi partition, collects the results and, finally, calculates the new codebook alone. At the beginning of the new iteration, the new codebook is distributed, in broadcast, to all of the slaves.

Fig. 3 is a screen-shot taken from XPVM. It reports a part of the diagram of the tasks activity versus the time. It refers to a run of the parallel LBG on 4 hosts, with an input data set of 16384 16-dimensional vectors and a codebook of 128 elements. The input data set was obtained from the image of Lena [22] of 512×512 pixels at 256 grey levels. It was divided into blocks of 4×4 pixels, and, in this way, 16384 16-dimensional vectors were generated. The figure illustrates the periods of activity and inactivity of the tasks, the exchange of the messages and the overhead introduced by their transmission. XPVM does not make the graphic tracing of the broadcast messages [5], that in the algorithm, are the messages relating to the transmission of the codebook from the master to the slaves. The scale of times was divided into parts labeled as (a), (b), (c) and (d); the operations executed during each of them can be summarized as follows.

(a) Preliminary phase. The master spawns the slaves and waits until all of them start correctly.

(b) Distribution of the input data set. In this phase, the master distributes (just once), to each slave, the portion of input patterns for which it has to

[5] PVM provides a function of sending data in broadcast to all the tasks; however, this is not a real broadcast function because it is implemented by sending the same copy of the data to all the tasks, one at a time.

calculate the nearest codeword.

(c) Voronoi partition calculation. This is the heart of the parallel algorithm. The first operation in this phase is the distribution, in broadcast, of the codebook from the master to all of the slaves. Even if we cannot see the graphic tracing of the broadcast message, we can see that, for the considered example, the transmission of the codebook introduces a negligible delay with respect to the whole phase. In fact, by carefully looking at all segments labeled as (c), we see that all of the slaves, practically, start working at the beginning of the phase. This is not true for the first iteration (the one after segment (b)), but, from the same picture, we can see that such a delay is the consequence of the messages transmitted during phase (b) still occupying the network. Phase (c) ends when the master completes the calculation of its quota of the Voronoi partition.

(d) Collection of the results from the slaves and calculation of the new codebook. This is the critical part for the parallelization; in fact, during its execution, only the master works, while the slaves, after they have sent their results, wait for the new codebook from the master. Therefore, it is opportune to try to minimize its duration. Several tests of the algorithm have been effected when m, N_P, N_C and N change and the results have been analyzed, both the graphic ones from XPVM, and the numerical ones from the profiling of the master. The authours concluded that, for the implementation of the parallel LBG on the MULTISOFT Machine, it is better to minimize the information transmitted on the network, but, in any case, not loading the master with the calculation of the Euclidean distance. This is calculated by the master only during phase (c) when it determines its portion of the Voronoi partition. According to these considerations, the information transmitted from a slave to the master is constituted, for each pattern, by the index of the nearest codeword and by the QE related to it. Each slave sends these results when it completes the calculation of its portion of the Voronoi partition. When all of these values have been collected by the master, it is able to calculate the total distortion and the new codebook. The master cannot calculate the new codebook until all of the data are received. In order to minimize its periods of inactivity, it operates as follows: it checks whether the results from any slave have been received and executes all of the operations it can execute with those data; after, it checks whether there are other results, and so on until it receives all of them. From the profiling of the master, effected for $N = 1, 2, 4, 8, 16$, the authors saw that the time it employs to calculate the new codebook, after all the results have been collected, is less than 1%; so, it can be considered negligible.

Fig. 4 refers to the same input data set employed for the example of Fig. 3 and, in this case, we have $N_C = 256$ and $N = 16$. We can see that the periods of inactivity of the tasks increase when N increases, too. This is due to the

greater quantity of data to be transmitted over the network. Therefore, when N reaches a certain value, the adding of other processors does not produce any benefit, i.e. the speed-up saturates.

4.2 ELBG

As we previously said, the implementation of the parallel ELBG is substantially identical to the implementation of LBG with the addition of the ELBG block. It is executed, in serial mode, only by the master.

Fig. 5 reports a part of the diagram of the activity of the tasks versus the time in the case of ELBG. The problem is the same as the one we considered in Fig. 3 for LBG. The meaning of segments (a), (b), (c) and (d) is the same as in Fig. 3 with the only difference that, in this case, inside phase (d), before the new codebook calculation, the master also executes the ELBG block. In order to visualize a greater number of iterations, in Fig. 5, a different scale for the axis of the times with respect to Fig. 3 has been chosen. However, we can take as reference the time of "Computing" of the slaves at each iteration; it is the same both in LBG and in ELBG.

The first difference we can notice between Figs. 3 and 5 is that, in the latter, the periods of inactivity of the slaves are longer than in the former. This has to be attributed to the execution of the ELBG block by the master; such an operation makes segments (d) longer than in LBG. The authors verified that the periods of inactivity are reduced when the complexity of the problem increases. In the cases analyzed, an increase in the complexity consists in the increase of N_C, N_P being fixed. Besides, as we can see in Fig. 5, the length of segments (d) is not fixed because of the ELBG block, whose computing time is not deterministic as in the calculation of the Voronoi partition.

Another difference between the diagrams that, in the authors' opinion, could be imputable to the pair PVM-TCP/IP, is related to segments (c). Both LBG and ELBG transmit the new codebook to the slaves at the beginning of (c) and, as the dimensions of the codebook are the same in both of the cases, the quantity of data to send over the network is the same. However, the time required for the transmission is negligible in Fig. 3, while it is not in Fig. 5. The runs were repeated several times, but the result did not change. In the authors' opinion, this could derive from different behaviour of the pair PVM-TCP/IP in the two situations examined. As a consequence of this, in ELBG, the period of inactivity of the master at the beginning of segments (d) is longer than in LBG.

5 Results

In this section the performances of parallel LBG and ELBG in terms of speed-ups (S) are reported. Performances, in terms of final MQE and number of iterations required for the convergence, are identical to the ones obtained by the corresponding serial versions. On this subject, several comparisons with other existing techniques [23,24,12,25], both hard and fuzzy, both K-means and competitive learning, are reported in [15,13,14]. In such comparisons, ELBG obtained results better than or equal to all of the other considered techniques, as regards both the final MQE and the number of required iterations. Besides, the difference in favour of ELBG increases when the complexity of the clustering problem increases, too and the final MQE ELBG obtains is, practically, independent of the initial conditions.

Before reporting the numeric values related to S, some considerations about the validity of the results obtained regarding the times of calculation are worthwhile. LBG is a deterministic algorithm, i.e., after the initial codebook has been fixed, it always develops in the same way. The same deterministic behaviour has to be followed both by the serial and by the parallel implementation of LBG. However, the utilization of a Fast Ethernet-based network, whose management is not deterministic, produced slightly variable results. Regarding ELBG, we must also consider the non-deterministic behaviour of the ELBG block. So, in order to make allowance for these factors, all of the results presented are the mean value of 20 runs. In Tab. 1 the performances of LBG and ELBG are reported in terms of S of the time required per iteration; the input data set is the same adopted in the previous section. Some considerations regarding these results follow.

- We can see that, for all of the examined cases, LBG presents a higher value of S than ELBG. This is obvious because of the serial execution of the ELBG block only by the master, while the time of activity for the slaves is the same as LBG.
- The number of processors (N) being equal, generally, S increases when the complexity of the problem increases, too. In this case, the complexity is determined by the dimensions of the codebook (N_C) because input data set is fixed. Such a trend is justified by the decrease in the time of inactivity of the slaves. However, there are some cases where this trend is inverted. For example, let us look at LBG when we change $N_C = 32$ into $N_C = 64$ for $N = 4$, $N_C = 64$ into $N_C = 128$ for $N = 8$, $N_C = 128$ into $N_C = 256$ for $N = 16$. The authors saw that such behaviour derives from the increase of the overhead in the transmission of the codebook from the master to the slaves.
- N_C being fixed, when N increases, S saturates. This occurs because the increase of N prolongs the periods of inactivity of the slaves and, in practice,

their contribution to the algorithm no longer increases. Both for LBG and ELBG, when N_C increases, saturation occurs for higher values of N. Also in this case, it is a good trend because both of the parallel algorithms considered perform better with high complexity problems.

6 Conclusions and future developments

The work presented in this paper is a preliminary study regarding the parallelization of algorithms for unsupervised learning on the MULTISOFT machine, a commodity supercomputer. In particular, two techniques for VQ have been analyzed: LBG and ELBG. We could see that some points, such as the calculation of the Voronoi partition, are easy to implement on the system described, others can be improved. Regarding ELBG, the authors are thinking of developing a new version of the ELBG block that could be executed in parallel by all of the hosts in order to increase the speed up of the whole algorithm. Another improvement, whose benefits would interest both LBG and ELBG, can be introduced in the inter-process communication by the utilization of a real broadcast function. In fact, in the current implementation of the two algorithms, the broadcast transmission of the codebook from the master to the slaves is effected by PVM by simply sending the same copy of the data, one at a time. With this policy of broadcasting, the speed-up tends to decrease when the number of PEs increases a lot. However, for the number of PEs considered in this paper, such behaviour does not take place. With a function of real broadcast (as the one provided by the UDP on a Fast Ethernet network), the master could transmit the codebook on the network, just once per iteration, and all of the slaves could read it at the same time making the speed-up higher than at present. The last improvement regards the possibility to use efficiently a distributed system whose PEs are based on processors of different speed and/or types, as the MULTISOFT machine is.

References

[1] W.Pedrycz, "Conditional Fuzzy Clustering in the Design of Radial Basis Function Neural Networks," *IEEE Transaction on Neural Networks*, vol. 9, pp. 601–612, July 1998.

[2] K.Fukunaga, *Introduction to Statistical Pattern Recognition*. 24–28 Oval Road, London NW1 7DX: Academic Press Limited, second ed., 1990.

[3] J.M.Jolion, P.Meer, and S.Bataouche, "Robust Clustering with Applications in Computer Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 791–802, Aug. 1991.

[4] N.R.Pal, J.C.Bezdek, and R.J.Hathaway, "Sequential Competitive Learning and the Fuzzy c-Means Clustering Algorithms," *Neural Networks*, vol. 9, no. 5, pp. 787–796, 1996.

[5] V. S. Cherkassky and F. M. Mulier, *Learning from Data: Concepts, Theory and Methods*. John Wiley and Sons, 1998.

[6] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4–37, 2000.

[7] K.K.Paliwal and B.S.Atal, "Efficient Vector Quantization of LPC Parameters at 24 Bits/Frame," *IEEE Transactions Speech And Audio Processing*, vol. 1, no. 1, pp. 3–14, 1993.

[8] P.C.Cosman, R.M.Gray, and M.Vetterli, "Vector Quantization of Image Subbands: A Survey," *IEEE Transactions on Image Processing*, vol. 5, no. 2, pp. 202–225, 1996.

[9] J. McQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967.

[10] Y.Linde, A.Buzo, and R.M.Gray, "An Algorithm for Vector Quantizer Design," *IEEE Transaction on Communications*, vol. 28, pp. 84–94, Jan. 1980.

[11] J.C.Bezdek, "Pattern Recognition with Fuzzy Objective Function Algorithms," in *New York: Plenum*, 1981.

[12] D.Lee, S.Baek, and K.Sung, "Modified K-means Algorithm for Vector Quantizer Design," *IEEE Signal Processing Letters*, vol. 4, pp. 2–4, Jan. 1997.

[13] M.Russo and G.Patanè, "Improving the LBG Algorithm," in *Proc. of IWANN'99* (J.Mira and J.V.Sánchez-Andrés, eds.), vol. 1606 of *Lecture Notes in Computer Science*, (Barcelona, Spain), pp. 621–630, Springer, June 1999.

[14] G.Patanè and M.Russo, "Comparisons between Fuzzy and Hard Clustering Techniques," in *Proceedings of WILF'99*, June 1999. in press.

[15] M.Russo and G.Patanè, "The Enhanced LBG Algorithm," *Neural Networks*, in press.

[16] G.Patanè and M.Russo, "Elbg implementation," *International Journal of Knowledge based Intelligent Engineering Systems*, in press.

[17] N.K.Ratha, A.K.Jain, and M.J.Chung, "Clustering using a coarse-grained parallel Genetic Algorithm: A Preliminary Study," in *IEEE Proc. of Computer Architecturs for Machine Perception*, 1995.

[18] E. Anderson, "The irises of the gaspe peninsula," *Bull. Amer. IRIS Soc.*, vol. 59, pp. 2–5, 1939.

[19] F.Ancona, S.Rovetta, and R.Zunino, "Parallel Architectures for Vector Quantization," in *Proc. of IEEE Int. Conf. on Neural Networks'97*, vol. 2, (Texas, Houston), pp. 899–903, June 1997.

[20] F.H.Bennet III, J.R.Koza, J.Shipman, and O.Stiffelman, "Building a Parallel Computer System for $18,000 that performs a Half Peta-Flop per Day," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '99)*, 1999.

[21] S.P.Lloyd, "Least Squares Quantization in PCM's." Bell Telephone Laboratories Paper, Murray Hill, 1957.

[22] D.C.Munson, Jr., "A Note on Lena," *IEEE Transactions on Image Processing*, vol. 5, p. 3, Jan. 1996.

[23] C.Chinrungrueng and C.H. Séquin, "Optimal adaptive K-Means Algorithm with Dynamic Adjustament of Learning Rate," *IEEE Transaction on Neural Networks*, vol. 6, pp. 157–169, Jan. 1995.

[24] B.Fritzke, "The LBG-U Method for Vector Quantization – an Improvement Over LBG Inspired from Neural Network," *Neural Processing Letters*, vol. 5, no. 1, pp. 35–45, 1997.

[25] N.B.Karayiannis and Pin-I Pai, "Fuzzy Algorithms for Learning Vector Quantization," *IEEE Transaction on Neural Networks*, vol. 7, pp. 1196–1211, Sept. 1996.

List of Tables

List of Figures

		LBG		ELBG	
N_C	N	*T it. (s)*	S	*T it. (s)*	S
32	1	0.62	1.00	0.79	1.00
	2	0.48	1.30	0.64	1.23
	4	0.23	2.73	0.40	1.97
	8	0.16	3.89	0.33	2.40
	16	0.13	4.60	0.32	2.49
64	1	1.17	1.00	1.41	1.00
	2	0.76	1.54	1.18	1.19
	4	0.45	2.60	0.87	1.61
	8	0.24	4.89	0.75	1.88
	16	0.19	6.26	0.73	1.93
128	1	2.28	1.00	2.60	1.00
	2	1.31	1.73	1.85	1.40
	4	0.75	3.05	1.27	2.04
	8	0.49	4.69	1.00	2.61
	16	0.28	8.17	0.91	2.86
256	1	4.51	1.00	5.00	1.00
	2	2.44	1.85	3.17	1.58
	4	1.31	3.44	2.02	2.48
	8	0.77	5.83	1.48	3.39
	16	0.60	7.51	1.25	4.00
512	1	9.30	1.00	9.90	1.00
	2	4.77	1.95	5.77	1.72
	4	2.49	3.73	3.50	2.83
	8	1.37	6.79	2.37	4.18
	16	1.03	9.02	1.86	5.32
1024	1	18.56	1.00	20.47	1.00
	2	9.39	1.98	11.10	1.84
	4	4.77	3.89	6.94	2.95
	8	2.59	7.17	4.41	4.65
	16	1.65	11.24	3.22	6.35

Table 1
Speed up (S) for LBG and ELBG

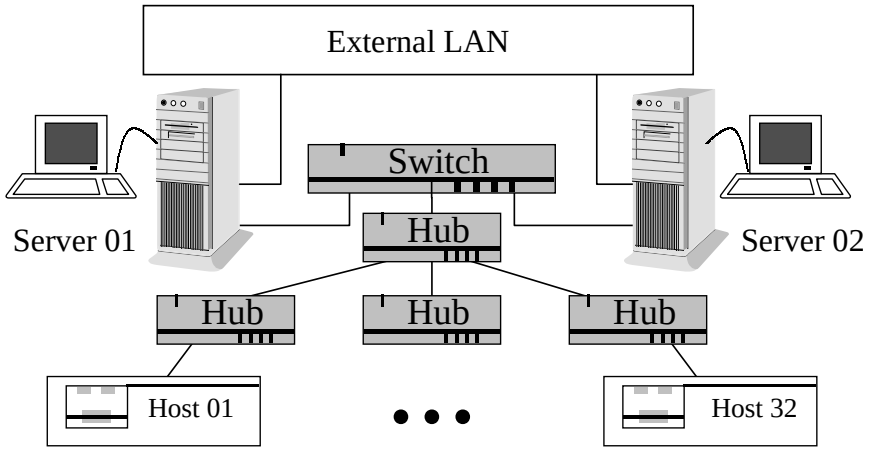

Fig. 1. The structural of the MULTISOFT machine

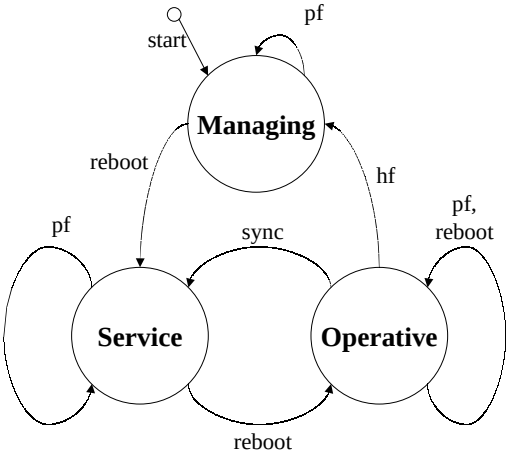

Fig. 2. The FSM describing the OS organization of the MULTISOFT machine

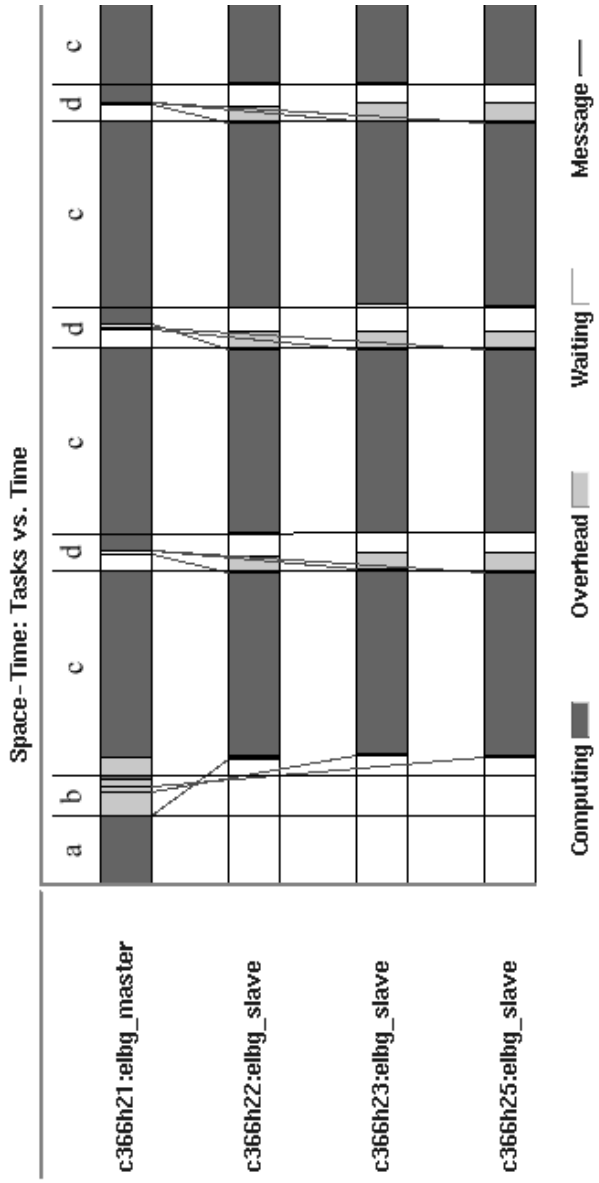

Fig. 3. Task vs. Time diagram for LBG with $k = 16$, $N_P = 16384$, $N_C = 128$, $N = 4$. The scale of times is such that an iteration is about 0.75 s.

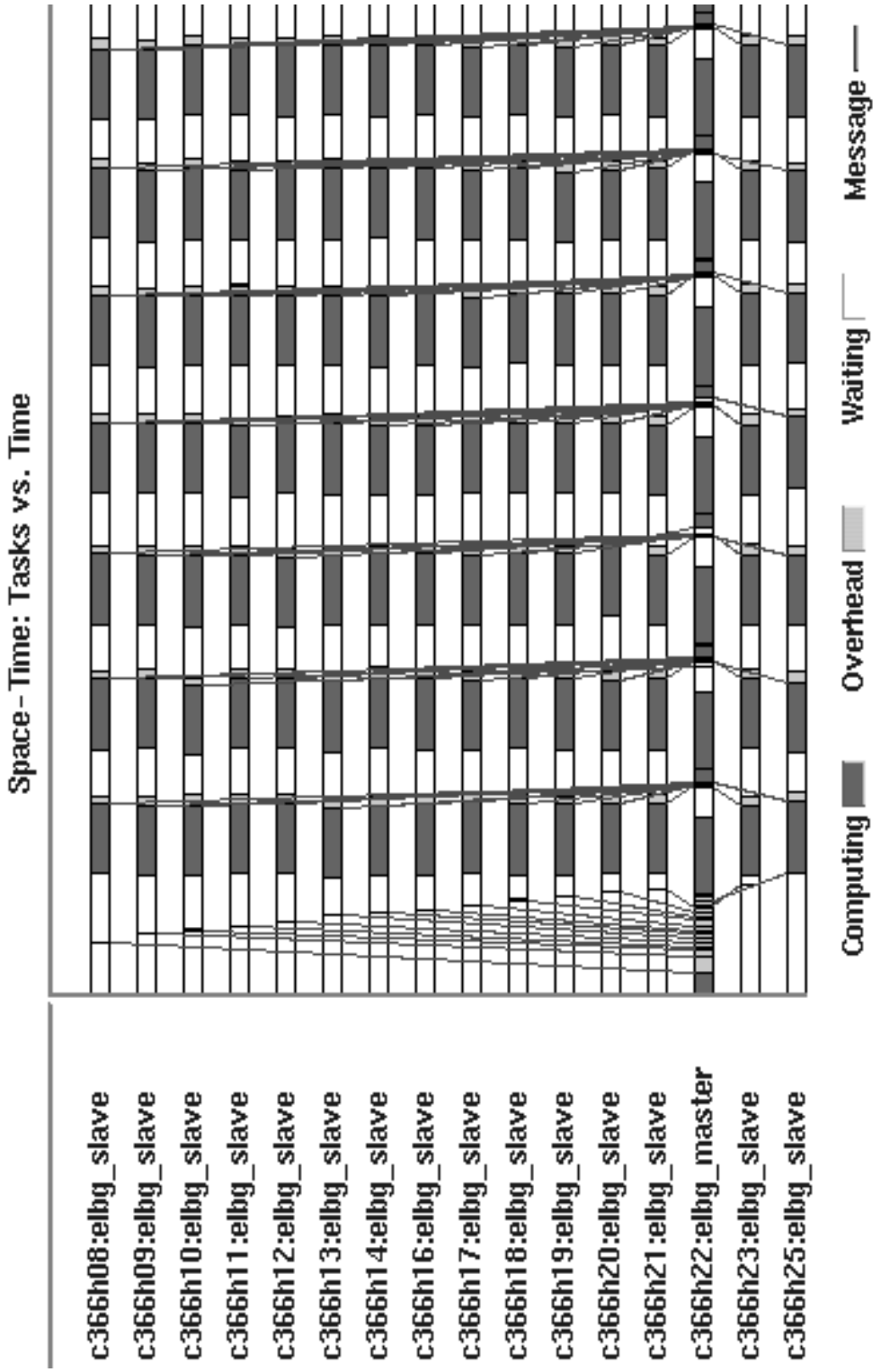

Fig. 4. Task vs. Time diagram for LBG with $k = 16$, $N_P = 16384$, $N_C = 256$, $N = 16$. The scale of times is such that an iteration is about 0.60 s.

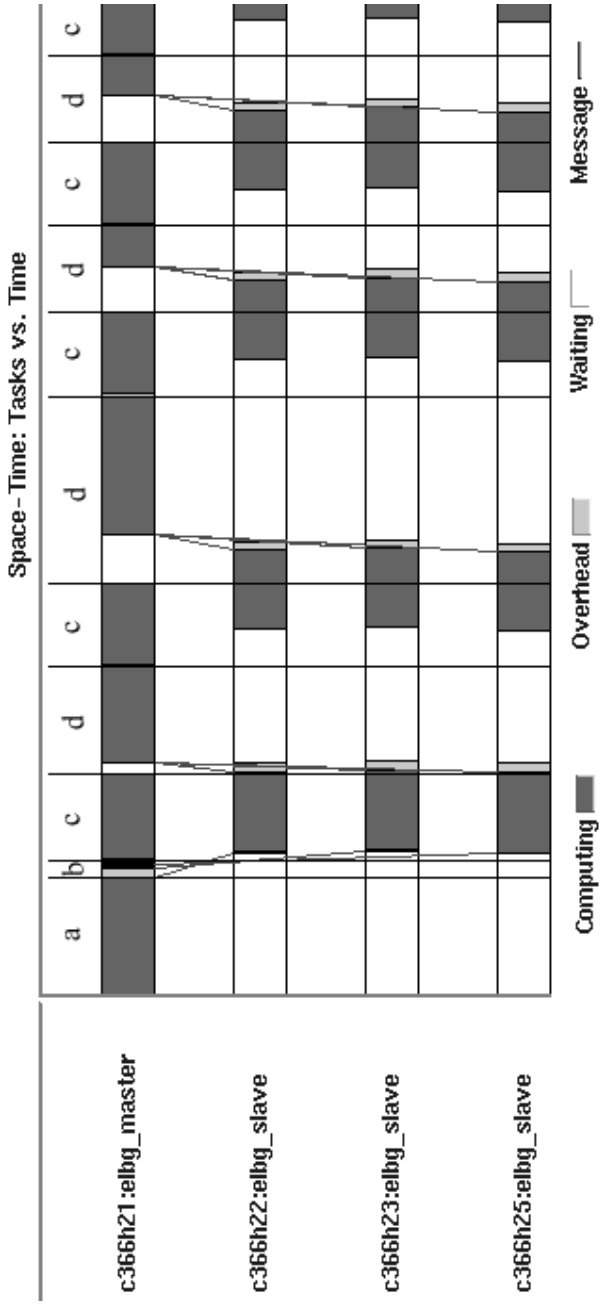

Fig. 5. Task vs. Time diagram for ELBG with $k = 16$, $N_P = 16384$, $N_C = 128$, $N = 4$. The scale of times is such that an iteration is is about 1.27 s.