

# Balanced workload distribution on a multi-processor cluster

J.L. Bosque<sup>a</sup>, B. Moreno<sup>b</sup>, L. Pastor<sup>b</sup>

*<sup>a</sup>Departamento de Automática, Escuela Universitaria Politécnica de la Universidad de Alcalá, Alcalá de Henares, Madrid, Spain*

*E-mail: jbosque@aut.alcala.es*

*<sup>b</sup>Departamento de Tecnología Fotónica, Facultad de Informática, Universidad Politécnica de Madrid, 28660 Boadilla del Monte, Spain. E-mail: lpastor@fi.upm.es, bmoreno@sidra.dtf.fi.upm.es*

## Abstract

This paper presents LoadBalancer, an application that aims to execute compute-intensive tasks over a cluster of machines linked through a local or wide area network. Workpackages are evenly distributed among the computers that compose the *virtual machine*. The available processors are arranged using a “farm” strategy, in which a master process sends workpackages to the slave processors that are integrated on the cluster. LoadBalancer has been developed for Construcciones Aeronáuticas, S.A. (Space Division), under the EU ESPRIT Programme.

## 1 Introduction

Parallel processing and multiprocessor machines have been posed as a solution for making high-performance computing both available and affordable for many scientific and engineering problems[1][2]. Nevertheless, parallel machines beyond shared memory multiprocessors using a few CPU's present higher prices for their hardware and software, requiring often skills that are out of many users' background. These reasons have prevented or at least constrained their widespread use in companies and research centres.

On the other hand, it is very common to find users whose computing needs have been solved during time by purchasing workstations that were interconnected later on by a relatively fast network. In a logical evolution, distributed systems [3] [10] have naturally appeared as a low-cost alternative for users wishing to achieve high computing power for solving their problems at an affordable cost. The main idea behind is to configure a group or cluster of workstations interconnected through a network to form a parallel virtual machine with a high computing power, able to solve compute-intensive problems in short times, using hardware resources that often

## 114 High Performance Computing

are already available. This has been made possible thanks to the speed, price and reliability improvements achieved by computer networks, in particular with the arrival of fiber optics.

A cluster or parallel virtual machine can be seen as a set of possibly heterogeneous, independent machines, connected through a fast communication network, working together under the management of a distributed software on the solution of a particular problem. The data communication and process synchronisation is performed using message-passing primitives, usually under a client/server architecture.

This paper presents LoadBalancer, a distributed application, implemented with a master/slave architecture under PVM (Parallel Virtual Machine), with the following objectives:

- To execute compute-intensive applications over a cluster of machines linked by an interconnection network, performing at the same time a balanced workload distribution among the heterogeneous set of processors which compose the cluster.
- To keep the communication overhead associated to the work distribution as low as possible (communication overheads affect very strongly multiprocessors performance).
- To decrease the overall system latency (the user response time from the instant when the execution is started to the moment when the results are produced).

LoadBalancer has been developed for Construcciones Aeronauticas, S.A. (Space Division) within the framework of the EU ESPRIT programme, focusing on the development of parallel Montecarlo methods for structure analysis.

The following sections describe the application environment and structure, the tests performed and the results achieved. Last, the conclusions that can be taken from the experimental results are presented.

## 2 Application description

### 2.1 Environment

The hardware over which the application runs is composed by a set of independent nodes, interconnected through a communication network. The nodes can have heterogeneous architectures, although all of them have to run under the UNIX operating system [6][9]. Therefore, the hardware can be seen as a distributed system [7].

The communication network used for linking the nodes can be local or wide area (the communication network can be also heterogeneous). An important aspect to take into account is the network traffic: heavily loaded networks can become a bottleneck, determining largely the overall application performance.

The hardware used is conceptually similar to a distributed memory multiprocessor. We will refer to it on the rest of the paper as the virtual machine (VM).

## 2.2 System configuration

The VM configuration is done dynamically. It can therefore be changed between different applications' executions. This process is done in a transparent way from the user point of view: the user only needs to provide a configuration file with the IP addresses of all the machines that can take part in the VM.

The application starts by reading the configuration file on a first machine, attempting later on the connection to the specified computers. If the connection process succeeds, the remote node is added to the VM. Otherwise the user is informed of the resulting error, being the operation continued with the remaining machines. This process is performed using PVM primitives [9].

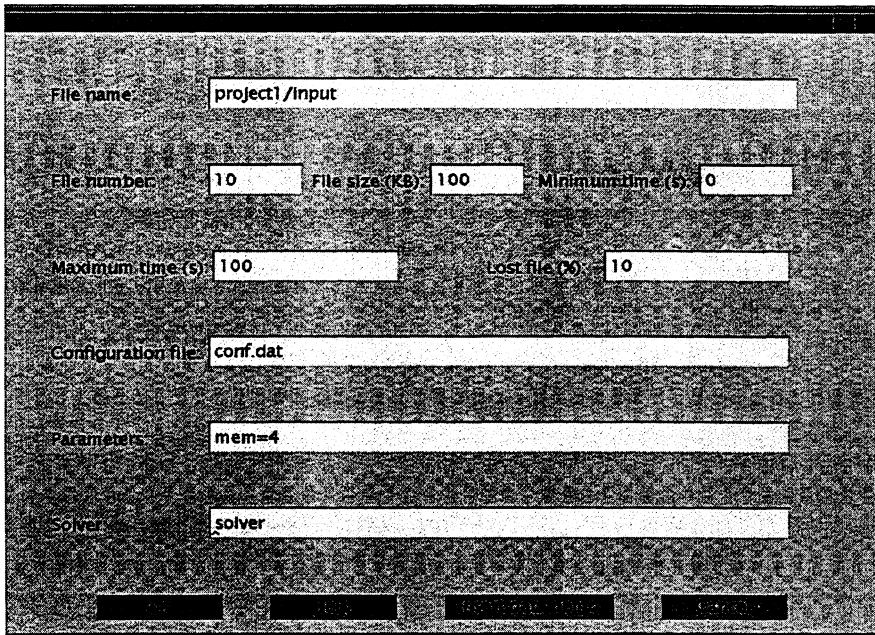
Once the final VM configuration is achieved, the user is presented with a graphical schematic describing the system configuration.

## 2.3 Application structure

The application is basically composed of a computing process, called 'solver', which has to process a (large) number of files. As stated on the introduction, the first objective posed for LoadBalancer is the even workload distribution among the available processors. For that purpose, a "farm" [7] strategy was selected: a master process is executed on a central node, being in charge both with the configuration of the VM and with the distribution of the work packages among the different slave nodes.

The master process has to perform a number of steps before the solver can start processing each of the data files associated to each run: first, a number of user-defined parameters have to be read in order to set up the application environment. Figure 1 presents a Motif window [9][10][11] showing the required data. After data is read, the master configures the VM using the IP addresses provided by the user.

## 116 High Performance Computing



The image shows a graphical user interface for setting application parameters. It consists of several labeled input fields arranged in a form. The labels are on the left, and the input values are in text boxes on the right. The parameters are: File name (project1 /input), File number (10), File size (KB) (100), Minimum time (s) (0), Maximum time (s) (100), Lost file (%) (10), Configuration file (conf.dat), Parameters (mem=4), and Solver (solver). At the bottom, there are four buttons: Run, Cancel, Help, and OK.

|                    |                 |                  |     |
|--------------------|-----------------|------------------|-----|
| File name          | project1 /input |                  |     |
| File number        | 10              | File size (KB)   | 100 |
|                    |                 | Minimum time (s) | 0   |
| Maximum time (s)   | 100             | Lost file (%)    | 10  |
| Configuration file | conf.dat        |                  |     |
| Parameters         | mem=4           |                  |     |
| Solver             | solver          |                  |     |
| Run Cancel Help OK |                 |                  |     |

Figure1 : User defined parameters for the application setup.

The third step performed by the master is the execution of the slave processes on each node, which include different solver instances. The master has to supply each slave with the execution parameters required by the solver as well as with the raw data files to be processed, waiting then to gather the results provided by each slave.

Slave processes, on the other hand, have to store the received file on the local node, start the solver execution using the data contained on the file and return the results produced to the master process. During the solver execution, each slave process has to check the execution time, aborting the solver if the time exceeds a predetermined span.

Last, the master has to gather the results provided by each slave on each of the allocated raw data files, storing them on a results data base. During the whole process, the master presents the user real-time graphics describing the application execution.

Once all of the files have been processed, the final statistics are computed, an accounting file summarizing the whole process is generated, and the application is finished.

LoadBalancer can be used with different solvers, keeping the processing structure independent on the data processing algorithms. In fact, it could be used with any application that performs heavy computation on blocks of data stored in registers.

Figure 2 describes the general application structure.

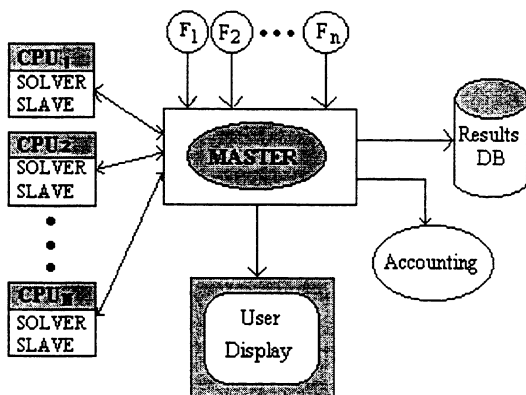


Figure 2: General application structure.

The graphical information presented by the master during the execution allows the user to find out the structure of the virtual machine (specifying whether the nodes are active, not active or communicating with the master) as well as the charge of work supported by each CPU from the beginning of the application until every moment, and the communication mean time between each host and the master. Figure 3 displays the way this information is presented to the user.

### 3. Experimental results

A number of tests have been performed to check LoadBalancer's performance when clusters and problems of different size are taken. This section presents first the experimental setup (including both hardware and software), describing afterwards the execution times obtained during the trials.

## 118 High Performance Computing

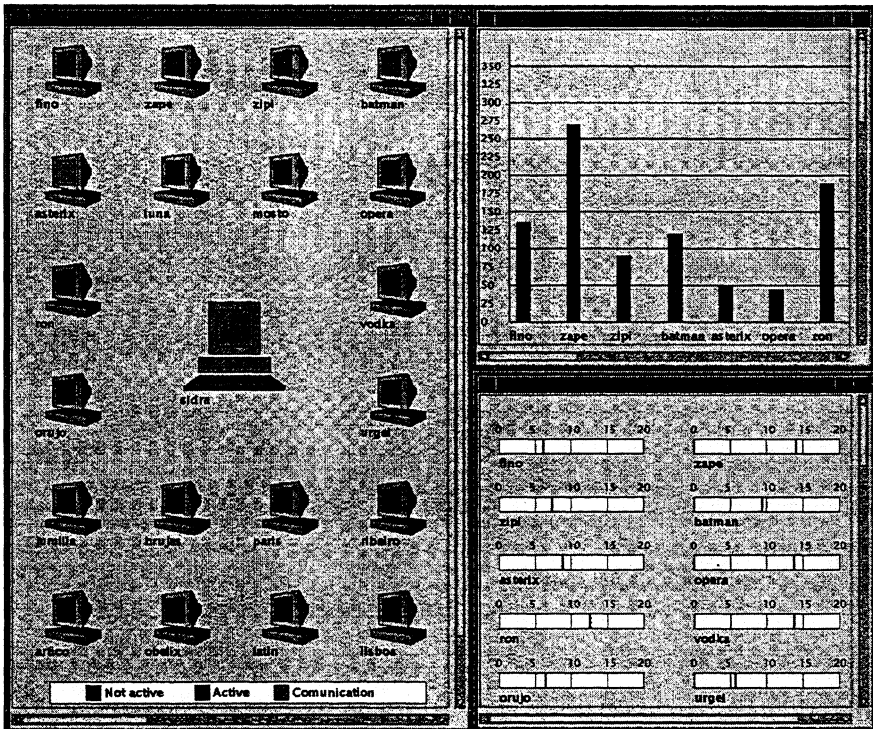


Figure 3 : Real time execution information displayed by the master process.

### 3.1 Hardware and software setup

The hardware available for testing the application consisted on nine ALPHA 400 workstations from DEC. One of the workstations is a server, being the machine

selected both for the execution of the solver when only one processor was used and for running the master process when more than one processor was used.

The other eight workstations were selected for the execution of slave processes. The ALPHA workstations' most salient features are:

- Server:

Processor: AS400 at 144 MHz

Memory: 64 MB

Mass storage: 2.5 GB on 1 SCSI disk

Operating system: DEC/OSF1 v3.2 (UNIX)

- Slaves:

Processor: AS400 at 100 MHz  
Memory: 32 MB  
Mass storage: 1.2 GB on 1 SCSI disk  
Operating system: DEC/OSF1 v3.2 (UNIX)

The available workstations are linked through a departmental LAN, belonging to the Laboratory of Telematics of the University of Alcalá de Henares (Laboratory of Telematics, Dept. of Automática, Univ. o Alcalá de Henares).

The fact that a departmental network has been used is relevant for two reasons:

- The situation is closer to "real world" working conditions.
- The LAN traffic conditions can affect differently subsequent executions, introducing a small degree of distortion on the times reported on this paper.

The LAN used is an ETHERNET using TCP/IP protocols. The network is decomposed on four segments, having a 16 input hub available to perform efficient routing. The network bandwidth is 10 Mbits/sg.

With respect to software considerations, it was mentioned before that LoadBalancer can work with different solvers. Although the application was developed within a structure analysis environment, the experiments presented here have used a simple matrix multiplication solver. Therefore, each of the input data files used for the tests contains two matrices and their respective dimensions.

Three different trials will be presented here. They involve processing three sets of 50, 75 and 100 files, having each file a random problem dimension (the matrices' dimensions, although compatible for matrix product, are selected randomly between a minimum value of 20 and a maximum of 500). For each of these trials different executions have been done, changing the number of processors while keeping constant the input data files. It has to be noted that the figures given for executions using only one processor have been obtained using an entirely sequential algorithm (only the solver was started on the server, having therefore no parallelism or communications overheads).

### 3.2 Execution times

The experimental results obtained with the hardware and software setup are summarized on figures 4 to 7.

Figure 4 gives the execution time dependence on the number of available slave processors (the figures do not include the master processors). Three problem sizes have been considered: the input data set was composed of 50, 75 and 100 files



## 120 High Performance Computing

respectively. Times given in figure 4 are total user response times. The time needed by the user to enter the input data has not been taken into consideration for these latency values, although the times needed for the configuration of the VM has been included. Figures 5 and 6 show the speedup and efficiency factors [4] [5] for processing 50, 75 or 100 files when one to eight slave machines are used.

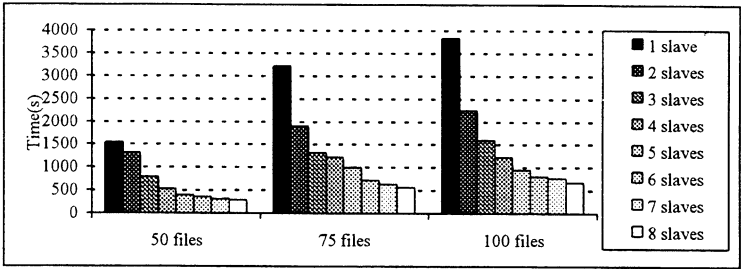


Figure 4: Execution time versus number of slave machines for different numbers of processed files.

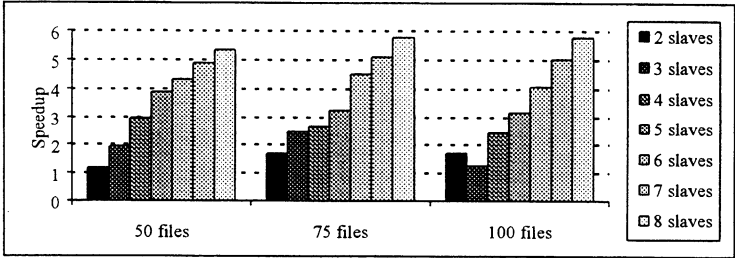


Figure 5 : Speedup figures for different VM configurations and number of processed files.

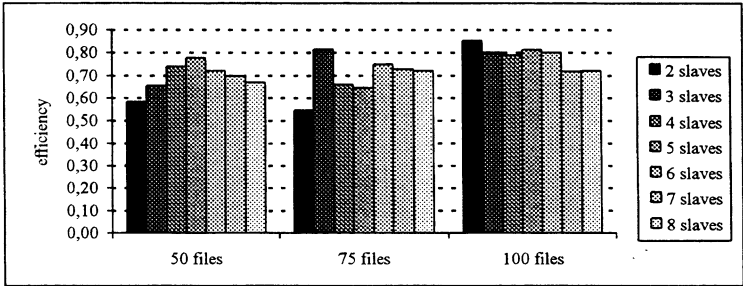


Figure 6: Efficiency figures for different VM configurations and number of processed files.

Last, figure 7 shows the dependence of the communications overhead with problem size and the number of slave machines. This overhead is given by the ratio total communication time for “n” slaves - total computation time for the same VM configuration :

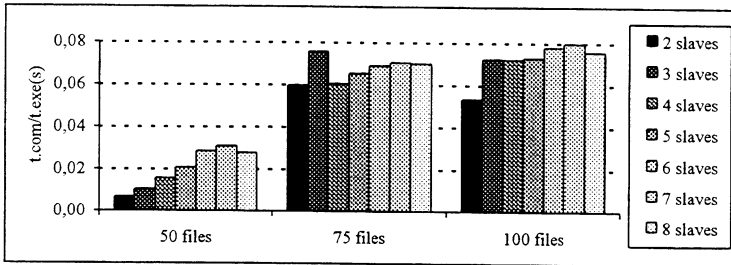


Figure 7: Dependence of communications overhead with problem size and numbers of slave machines.

## 4 Conclusions

The analysis of the experimental results allows the formulation of a number of conclusions :

First, the exploitation of asynchronous communication protocols such as the one implemented in LoadBalancer for the master/slaves communications allows the achievement of low communication overheads. As it can be seen in figure 7, These overheads have been always below the 10%, having reached an average value around 7 to 8%.

Second, the numbers obtained both for speedup and efficiency are quite good :for larger jobs, the efficiency keeps around or above 70%. For smaller jobs the initialization times affect negatively the application performance.

It has to be remembered that the values used for execution over only one processor include just the solver, without parallelism or communications overhead. Moreover, the machine used for these serial executions is the most powerful one, making the results look worse.

The application structure makes it also to reach a good scalability degree : increasing the number of slave processors from one to eight, for processing 100 files, makes the efficiency vary between 72% to 85%.

Last, it has to be noted that the results given in this paper have been obtained with a communication network shared with other users. Since the executions on just one processor do not use the network, these results could be further improved by restricting other user's network usage.



## 122 High Performance Computing

### 5 References

- [1]- Kevin Dowd, '*High Performance Computing*', Editorial O'Reilley & Associates, Inc. 1995.
- [2] - Bruce P. Lester, '*The Art of Parallel Programming*', Editorial Prentice- Hall International, 1994.
- [3] - Andrew S. Tamenbaun., '*Distributed Operating Systems*', Prentice-Hall 1996.
- [4] - Kai Hwang, '*Advanced Computer Architecture*', Mc Graw-Hill, 1993.
- [5] - V. de Carlini and U. Villano, '*Transputers and Parallel Architectures*', Ellis Morwood, 1991.
- [6] - Kay Robins & Steven Robbins, '*Practical UNIX Programming*', Prentice-Hall, 1996.
- [7] - G. Colouris, '*Distributed systems : Concepts and Decision*', Addison-Wesley, 1996.
- [8] - Shivarati et al. '*Load Distributing for Locally Distributed Systems*', (web).
- [9] - 'PVM 3 '*User's Guide and Reference Manual*', ORNL/TM-12187, May 1994.
- [10] - Open Software Foundation, '*OSF/Motif Style Guide*' for OSF/Motif Release 1.1, Prentice-Hall, 1991.
- [11] - Open Software Foundation, '*OSF/Motif Programmer's Guide*' for OSF/Motif Release 1.1, Prentice-Hall, 1991.
- [12] - Open Software Foundation, '*OSF/Motif Programmer's Reference*' for OSF/Motif Release 1.1, Prentice-Hall, 1991.