An Application Level Load Balancing Mechanism for Heterogeneous Clusters Programming*

David Sánchez, Elsa Mª. Macías and Álvaro Suárez
Grupo de Arquitectura y Concurrencia (GAC)
Departamento de Ingeniería Telemática (U.L.P.G.C.)
Campus Universitario de Tafira, 35017-Las Palmas de Gran Canaria, Spain
e-mail: {dsanchez,elsa,alvaro@cic.teleco.ulpgc.es}

Abstract. *Heterogeneous clusters provide an attractive scalability of low cost in terms of computation power. However, these systems are much complicated to program than dedicated parallel machines due to the perfomance of nodes and traffic in the network changes randomly. Execution of a parallel application in a heterogeneous cluster without keeping in mind these considerations, it can imply that some slow nodes can be overhead, while fast nodes can be in an idle state. Therefore, in order to avoid this situation, it is necessary to apply some scheme of load balancing. In this paper we present an application level load balancing mechanism for heterogeneous clusters.*

Keywords. Heterogeneous Clusters, Load Balancing, Execution Time.

1. Introduction

In the last years a technological growth related to high speed netwoks and computers has happened. Network of workstations provide an attractive scalability in terms of computation power and memory size. It is becoming strongly competitive compared to expensive parallel machines. Clusters [1] and Grid [2] computing are new computation ideas that take advantage of independent resources to build a unified resource for massive computation, distributed computation, data storing, etc.

Interest in heterogeneous computing systems continues to grow in the research community. However, there are many open problems [3] that are not solved, such as low overhead data transmission over heterogeneous networks, to improve communication latency, to investigate the best way to scale these environments, and so on.

Networks of workstations are much complicated to program than dedicated parallel machines. It is due to a multi-user environments, heterogeneous networks and different resources that can vary performance [4].

If we execute a parallel application in a network of workstations and we do not keep in mind communication latency and nodes performance, then the application execution time depends on the speed of the slowest node. This implies that some slow nodes can be overhead, while fast nodes can be in an idle state. Therefore, the execution time of each node must be balanced. For that reason, we need to apply some scheme of load balancing.

Load balancing implies to assign to each processor a load proportional to its performance, and so minimizing the execution time of the overall program, and preventing that some processors can be an idle state while others are in an overhead state. The minimal requirement in a load distribution scheme is to guarantee non-idleness during runtime [5].

Load balancing can be static (done at compile-time) or it may be dynamic (done at run-time) [4][6]. The distribution of tasks can be very arduous if processors have different speeds or memory resources, variable external load due to multiple users or an heterogeneous physical

*Research partially supported by Spanish CICYT under Contract: TIC2001-0956-C04-03.

layer network. Static balancing avoids overhead at run time. However, a dynamic load balancing is more appropriate in a network of workstations because of the performance of nodes and traffic in the network changes randomly. This kind of mechanism is crucial in clusters like [7] in which nodes can be added to share the workload dynamically. We have implemented a library named LAMGAC [8] to let the programmers to expand a network of workstations with wired or wireless computers (LAN-WLAN cluster) at run time of parallel applications.

In this paper, we present an application level load balancing mechanism that deals with the problem of load balancing in heterogeneous clusters. In practice, the execution time of the parallel applications using this mechanism is in a high degree decreased.

The rest of the paper is organized as follows. In section 2 we review some aspects about heterogeneous clusters. In section 3 our load balancing mechanism is presented. Then, in section 4 two applications that use this mechanism of load balancing are presented. In section 5 we present some experimental results for the two above applications. Finally we sum up the conclusions and we present the future work.

2. Some Considerations about Heterogeneous Clusters

The set of computers available in a heterogeneous clusters can include a wide range of architecture types such as PC-based machines, high-performance workstations, shared-memory multiprocessors, etc. Despite the numerous difficulties caused by heterogeneity [5] distributed computing offers many advantages, such as low cost computing. On the other hand, resources of these environments take advantange of recent technologies.

In heterogeneous environments, nodes performance and data communications among processes are two dominant factors in the execution of parallel applications. Perfomance of a node depends of parameters related to characteristics of resources [4] such as CPU speed and memory size, and also, the load on current time (multi-user environment). Communication time depends of several factors [5] such as network bandwidth, network topology, size of communicated data, transmission and reception buffer size, etc. We consider both aspects in order to balance the load.

When we want to do load balancing, we can not act on nodes performance, because it is inherent to the programmer. However, if messages size are high, we can keep it on mind to control the communication time.

A message sent to another node may cause the initiating process to hang the send routine although a non blocking send mechanism is implemented. The reason can be found in the message buffer size on the receiving node. Dependent on the message size, network bandwidth and other user processes the routine delays the further execution more or less likely [5]. In a low performance systems and in applications which need high data transmission, communication time is dependent on the size of message being sent.

If we define parameter *r* as the relationship between the calculation time and the message size sent, then, applications can be classified in two ways:

a. Low size data communication and high calculation time $\rightarrow$ High ratio (r)

b. High size data communication and low calculation time $\rightarrow$ Low ratio (r)

$$r = \frac{calculation\_time(ms)}{message\_size(KB)}$$

In applications where the size of communicated data is high (r is low), it is necessary to consider the message size, because it can influence negatively in the communication time, and therefore, in the load balancing. However, in applications where *r* is high we can consider communication time constant because it has minimum influence due to the message size.

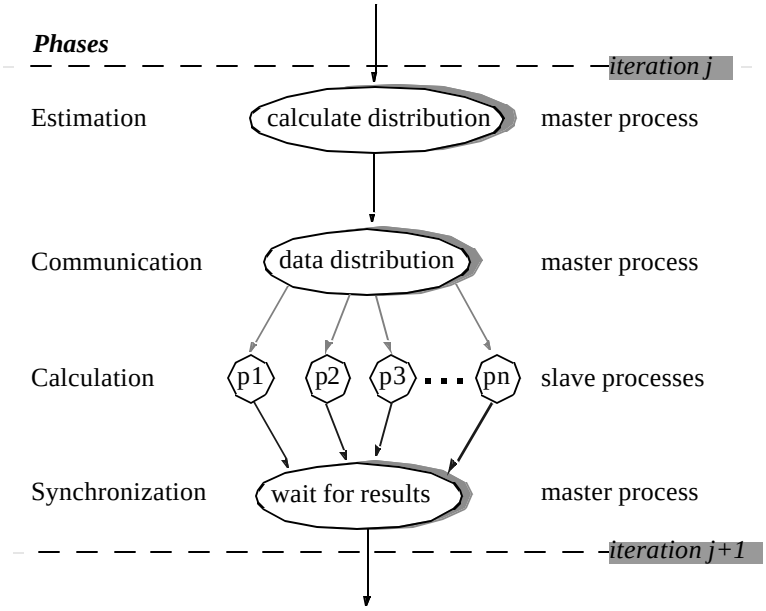

Fig. 1. Phases of the Load Balancing Mechanism

3. Load Balancing Mechanism

In this section we present a mechanism to balance the load in heterogeneous clusters. This scheme is divided in four phases and it is repeated periodically until there are not data to distribute (figure 1).

- *Estimation phase.* The program instance that executes the balancing mechanism (in master process in the parallel program) predicts the future performance for each process (slave processes) based on the very last previous information, this is, it calculates next data distribution to be sent to each slave process. Initially, it distributes the same amount of data among the processes.
- *Communication phase.* The estimated data distribution calculated in the previous phase is communicated to slave processes.
- *Calculation phase.* Each slave process makes computations with the data sent by the master process.
- *Synchronization phase.* Load balancing mechanism waits for all processes, this is, it receives of each process the results computed during the calculation phase.

For every parallel algorithm iteration, are mesaured the execution and computation times for each slave. Then, the master process makes estimations with these values and computes the next data distribution. Next, these measurements and estimations are presented:

- *Execution time of process i during the iteration j* ($t\_exe_j\ (pi)$). Total time it takes for a slave process *i* to receives data, to carry out the calculations and return the results. It is measured by load balancing mechanism.
- *Calculation time of process i during the iteration j* ($t\_calc_j\ (pi)$). Time it takes for a slave process *i* to carry out the calculations. It is measured by own process and it is communicated to the program instance that executes the balancing mechanism at the synchronization phase.
- *Communication time of process i during the iteration j* ($t\_com_j\ (pi)$). Time elapsed to send data and receive results, this is,

  $$t\_com_j\ (pi) = t\_exe_j\ (pi) - t\_calc_j\ (pi)$$

- *Information unit sent to process i during the iteration j* ($unit_j\ (pi)$). Minimum amount of data sent to compute a unit result. For example, for matrix multiplication, the information unit is equal to one row if master process distributes rows among slave processes.
- *Information unit time of process i during the iteration j* ($t\_unit_j\ (pi)$).

Average time spent on calculation phase by information unit.

$$t\_unit_j(pi) = \frac{t\_calc_j(pi)}{n^\circ unit_j(pi)}$$

For applications with high *r*, the load balancing mechanism estimates, in each iteration of the parallel application, the number of information units to be communicated to each process as:

$$n^\circ unit_{j+1}(pi) = \frac{t\_exemax_j - t\_com_j(pi)}{t\_unit_j(pi)} \quad \text{(Eq.1)}$$

where:

$$t\_exemax_j = max\{t\_exe_j(p1),\ldots,t\_exe_j(pn)\}$$

Due to *r* is high, then message size is similar among iterations and we can consider constant communication time from an iteration to another. In this way, we are balancing calculation time to the maximum one of each iteration.

For applications with low *r*, it can not be applied expression (Eq.1) because if the mechanism of load balancing sends different data size from an iteration to another, it causes different communication times. For that reason, we modify (Eq.1) such that information units number communicated to each process:

$$n^\circ unit_{j+1}(pi) = \frac{t\_exemax_j - \sum_{k=1}^{i-1} t\_com\_s_j(pk)}{t\_exe_j(pi)} \times unit_j(pi)$$

In this formula, $t\_com\_s_j\ (pk)$ is the time to send a message from program instance that executes the balancing mechanism to process *k*, and we are considering the communication time spent to send data to the slave processes minor to process *i*. In order to measure in a precise way $t\_com\_s_j\ (pk)$, the program that executes the load balancing mechanism has to implement blocking send routines in the distribution phase. The load balancing program instance communicates data of the slave process with more execution time, measured in the previous iteration of the parallel application, to the slave process with smaller execution time.

4. Application Examples

Our dynamic load balancing strategy has been tested for Master-Worker parallel applications. This is, there is one load balancing program instance (master process), and there are multiple slave processes. The objetive of the load balancing mechanism is to balance the load of each slave process to finish all of them at the same time. Slave process performs the computation task.

We have used two different applications for testing our scheme: a Hw/Sw Codesign Tool and a Matrix by Vector Multiplication.

4.1 Hw/Sw Codesign Application

Our research group has developed a Hw/Sw Codesign Tool named GACSYS [9][10]. The objetive of this tool is to optimize and satisfy a given set of design constraints. This tool is divided in two phases:

- *Parameter Estimation*. Hw/Sw parameter estimations are obtained for each process of the input data specification written in VHDL language such as execution time, power consumption and area. Depending of the used functional units number, this phase can be heavy computationally.
- *Partitioning*. An optimization algorithm based on Simulated Annealing finds a partition that satisfies the design constraints.

For testing our load balancing mechanism we have used the parameter estimation phase.

In the solution of this problem, the data distribution consists of sending to each slave process the number of combinations of functional units. The slave proceses will carry

out both parameters set calculation and number of combinations sent. During the synchronization phase of this application, the master process waits for the parameter estimation calculated in the slave processes.

4.2 Matriz by Vector Multiplication Application

Initially, the vector is broacasted to all slave processes. In the data distribution the load balancing mechanism sends a given number of rows of the matrix. Calculation phase consists of computing the row by vector multiplication as many times as rows have been communicated. The results of these multiplications are sent to the program instance process in the synchronization phase.

5. Experimental Results

In this section, we present performance results for our application level load balancing mechanism described in this paper. We have realized the experiments in a network of workstations with the specifications of table 1. The network bandwidth is 10 Mbps.

Table 1: Computing Resources Characteristics

Computer	Processor and Memory Size
PC1	Pentium III 733 Mhz 256 MBytes
PC2	Pentium III 733 Mhz 64 MBytes
PC3	Pentium 200 Mhz 64 MBytes
PC4	Pentium 133 Mhz 32 MBytes

We used C language and MPI library [11] to implement the applications described above. In order to obtain performance results we have implemented several experiments:

- Experiment one (E1) that consists of the sequential execution of the applications on the fastest processor (PC1).
- Experiment two (E2). The program that implements the load balancing runs on PC4 and slaves processes on the remainder computers.
- Experiment three (E3). Load balancing mechanism runs on PC1 and slaves processes on the remainder computers.
- Experiment four (E4). Load balancing program instance on PC4, but information unit is always equally distributed among slave processes (no load balancing).
- Experiment five (E5). Load balancing mechanism on PC1, but information unit is equally distributes among slave processes (no load balancing).

5.1 Hw/Sw Codesign

This application is appropriate to apply Eq.1, because communicated message size is similar from an iteration to another, and therefore, variations in the communication time does not depend of the bytes sent. Experiments realized for E2 to E5 have been executed for a initial distribution of information units equal to one, ten and twenty.

Figure 2 shows performance results for the differents experiments realized. Axis X represents initially distributed information units number, and the axis Y represents execution time in seconds.

We clearly observe that the execution time using our load balancing mechanism is better than the obtained results without load balancing strategies (about six times lower). However, the execution time without balancing is worse than sequential execution time due to synchronization phase in the parallel program is dominated by slow node.

Increasing initial data size, load balancing strategies spend a little more execution time due to the wait time on synchronization phase until achieve homogeneous execution time. Therefore, it is important initially to send few data in order to achieve balance time before.

Usually, an homogeneous execution time is obtained on third iteration of the parallel

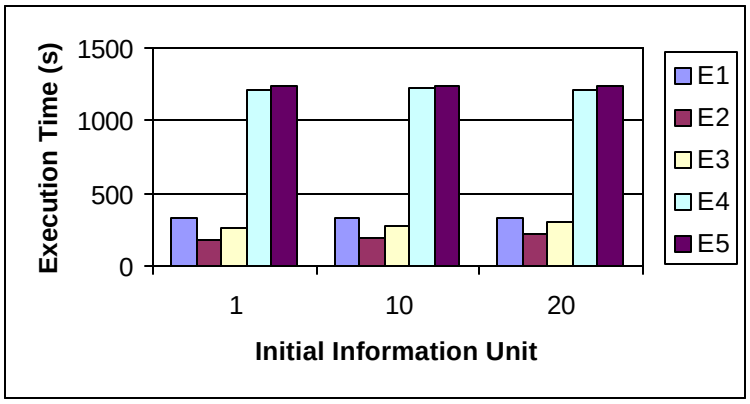

Fig. 2. Execution Time: GACSYS Hw/Sw Codesign Parallel Algorithm

algorithm.

5.2 MxV Multiplication

We have used for testing a matrix with 5000 rows by 1000 columns of integers. Due to little varations of execution time among iterations it can imply that the mechanism of the load balancing sends different rows number (different message size), therefore, for this application is suitable to apply the modified Eq.1.

Figure 3 shows performance results for the differents experiments realized, and these are the same implemented in the above application. We clearly observe that sequential execution is faster than execution of parallel application. It is due to in the parallel application the communication time is higher than calculation time. However, we can see that the execution time of the parallel application with load balancing mechanism is lightly smaller than execution time without load balancing mechanism.

Also, we can observe the influence of message size in the time used in the parallel application. Experiments E2 and E4 spend more execution time than the time for experiments E3 and E5, because the load balancing mechanism is running in a low performance node (PC4).

6. Conclusions and Future Work

In this paper, we present an application level load balancing mechanism for heteregeneous clusters. Such clusters can have different network topologies and computers with variable performance. In order to avoid this roughness,

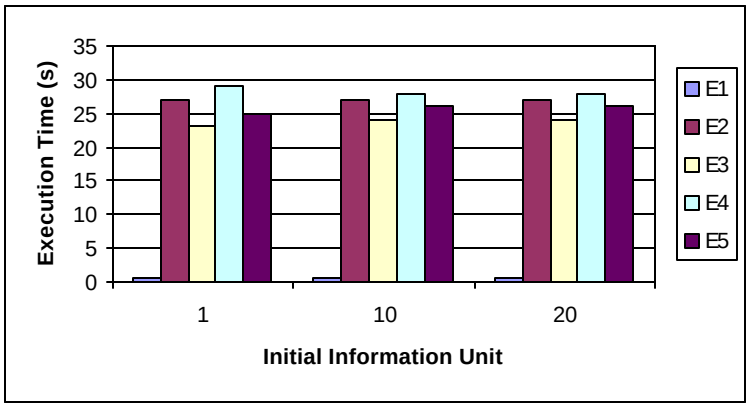

Fig. 3. Execution Time: MxV Multiplication Parallel Algorithm

we have propose two formules that we apply in accordance with calculation time and message size ratio. From the experiments, one can see clearly that load balancing is essential in parallel applications, resulting in optimal execution time, overalls in applications with low size data communication and high calculation time.

In our ongoing work we want to evaluate our load balancing mechanism in a hybrid network with wired and wireless links (LAN-WLAN cluster) using LAMGAC library. In the LAN-WLAN cluster, the application level load balancing mechanism not only has to deal with heterogeneous proccessing speeds but also with heterogeneous communication links and a dynamic number of computational resources. On the other hand, we want to modify the application level load balancing mechanism to consider one load balancing program instance on each process of the parallel program. It would be also interesting to ease the programming of the applications in this heterogeneous clusters, for which a middleware must be implemented. A comparison of both mechanisms must be done in order to decide what load balancing mechanism is more suitable.

References

[1] Mark, B. Cluster Computing White Paper. Final release. Version 2.0, 28th December 2000.

[2] I. Foster and C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann. 1999.

[3] T.D. Braun, H.J. Siegel, A.A. Maciejewski. Heterogeneous Computing: Goals, Methods, and Open Problems. Parallel and Distributed Processing, Techniques and Applications, vol I, pp. 7-18. Las Vegas, USA. June 2001.

[4] M. Zaki, W. Li, S. Parthasarathy. Customized Dynamic Load Balancing for a Network of Workstation. Proceedings of 5th High Performance Distributed Computing. Syracuse, NY, August 1996.

[5] Markus Fisher. Dynamic Load Balancing for Heterogeneous Parallel Enviroments. Paderborn, Germany. April 1997.

[6] Shahzad Malik. Dynamic Load Balancing in a Network of Workstations. 95.515F Research Report. November 2000.

[7] E. Macías, A. Suárez, C.N. Ojeda-Guerra, E. Robayna. Experimenting with the Implementation of Parallel Programs on a Communication Heterogeneous Cluster. Parallel and Distributed Processing, Techniques and Applications, vol II, pp. 829-835. Las Vegas, USA. June 2001.

[8] E. Macías, A. Suárez, C.N. Ojeda-Guerra, E. Robayna. Programming Parallel Applications with LAMGAC in a LAN-WLAN Environment. 8th European PVM/ MPI. LNCS 2131. Springer Verlag, pp 158-165. September 2001.

[9] J.P. Castellano, D. Sánchez, O. Cazorla, J. Bordón, A. Suárez. GACSYS: a VHDL-based Hw/Sw Codesign Tool. Design and Diagnostics of Electronic Circuits and Systems, pp 293-299. Szcyrk, Poland. September 1998.

[10] D. Sánchez, J.P. Castellano, A. Suárez. Hardware/Software Partitioning based on Simulated Annealing. Modeling and Simulation, pp 115-122. Las Palmas de Gran Canaria, Spain. September 2000.

[11] Information available in: http://www.mpi-forum.org