A Function to Dynamic Workload Allocation in Distributed Applications *

F. José Alfaro, José A. Gallud, José L. Sánchez

Universidad de Castilla-La Mancha, Escuela Politécnica
Campus Universitario, 02071 Albacete, Spain
{falfaro,jgallud,jsanchez}@info-ab.uclm.es

Abstract. A general goal of distributed systems is to reduce the global time of executing a parallel application. In this paper we have developed a distributed-system software monitor named Daem2 and a workload function that splits data among the processes. This function collects the information about load value from each computer, provided by Daem2 and then takes a decision. Finally, we show the comparative results between the uniform function and the dynamic one. Such results have been obtained running a numerical parallel program on a Linux workstation network.

1 Introduction

Recent development in parallel distributed systems and parallel programs have resulted in increased use of suitable environments like PVM [6] or MPI [14] which make the message-passing programming to solve complex problems easier and faster for users.

Massively parallel computers are a popular class of distributed memory systems that are intended to run large applications. Such computers are considered the most promising architecture to achieve teraflops computational power. Such systems must be programmed following the traditional paradigm for programming multicomputers or, in general, distributed-memory multiprocessors.

Both PVM and MPI frameworks allow the user to write his or her applications as a collection of cooperating tasks. A library of standard interface routines is used to bind a task with PVM or MPI functions. A number of differences can be found between both environments. PVM provides a virtual machine and MPI is intended to support the message-passing model. The results of this paper can be obtained with any of them, but we have focused on the MPI software.

In this paper we show both the direct effects and the side-effects of overheads computers in a distributed system and we define a function that takes the information about overloaded computers and then decides the workload allocation dynamically by data partitioning. For this purpose, we have designed a distributed-system software monitor named Daem2, which provides load information of each machine to the application. Moreover, we have developed our

* This work was supported in part by CICYT under Grant TIC94-0510-C02-02
experiences by applying the "crowd" computing model: a number of processes executing the same code with different data in order to solve a unique problem.

A special feature in this paper is the selection of the problem. We have chosen a numerical problem based on matrix multiplication. There are a lot of real applications that employ matrix multiplication in the process of solving a problem [7, 9].

The next section presents the definition of the function that decides the workload allocation of data. In section 3 the main aspects of the Daemon2 process, our software monitor, will be shown. In section 4 the results will be given and, finally, in section 5 we shall describe our conclusions and future work.

2 Dynamic load balance

If in massively parallel machines, like multicomputers and multiprocessors, a good distribution and load balance is important [1, 10, 13, 16, 17], in workstation network this is even more important [4, 5, 15]. These workstations are usually communicated by nets of greater length where conflicts are easier and faults appear, and so communication latencies are more important. As this is so important load delivery between different machines was done as exactly as possible, minimizing communications and maximizing machines throughput.

There are a large variety of parallel applications which follow the master-slave model [7, 9, 3]. The master process is in charge delivering work to the slave processes, which is processed by the later. The communication is traveling in both directions, from the master to the slaves distributing information, and from the slaves to the master collecting results. A variety on this scheme is that where the processes communicate between each other, for information interchange or for synchronization, but in any case there is always a load processing delivery between slaves.

In a workstation network one or more processes of the application would be placed on every machine we used. These machines can have different characteristics as for calculation speed, system load, utilization of their resources like memory, input / output, etc. Evidently with these characteristics the same quantity of work should not be given to each machine, but rather each machine should be allocated a quantity of work proportional to its capacity and state.

We have concentrated on evaluating the situation of each machine, which can be seen as the load that is supporting. This load would be composed of all that the machine is using, including the different processes, both from the application and from the machine itself, or even from other users. With all this we decided to see the CPU usage in each machine. In function of this information some load delivery would be taken, performing the distribution proportional to each machine load with respect to the global system load, which would be the partial load sum of each machine in the system. Therefore periodically the slave processes will evaluate the use that is being made of the machine in question and will inform the master process. This process will evaluate those pieces of partial information with respect to the total and the work delivery will be continued.
proportionally to the load that each machine is supporting, giving more work to those machines that are underemployed. This permits us to get the maximum efficiency from the less used machines.

To prove this we have used some parallel applications [2, 7]. In the section 4 we shall show the results obtained with a parallel version of a problem based on matrix multiplication using some machines by means of MPI. In this application we have a master process distributing the first matrix rows between the slave processes in order that these may carry out the multiplications in a distributed fashion and find out their corresponding result matrix rows, that return to the master process. Every slave process collects a block of rows, does the calculations to obtain their corresponding rows in the result matrix and return them to the master process. The quantity of rows the master gives to each process will be always the same in the uniform way. In the case of the dynamic function the number of rows will be proportional to the CPU available usage from each machine.

As we see in the next point we use a daemon process called Daem2 placed in every machine that we use, which collects the CPU usage information. This information is collected periodically by the slave processes that broadcast it to the master so that it may, in function of those utilization, carry out the distribution of rows. The results obtained in a dynamic delivery way will be compared in the section 4 with the corresponding uniform function, under the same load conditions.

3 A distributed-system software monitor

Given an initial design of a distributed system, an important consideration is load measurement of the system using concepts of performance system evaluation. If we wish obtain efficient executions in a given system, we have to take into account a number of parameters such as CPU utilization, system throughput, response time, disk utilization, memory usage, service time, and so on [12]. In this section we explain the design of our Daem2 process; an application that is able to measure performance of a distributed system.

In distributed computing other important issues must be added to the above: speed-up, fault tolerance, latency, network bandwidth, speed, etc. Monitoring a distributed system is more difficult than monitoring a centralized system.

A parallel application must be able to improve the global time (presence time) with regard to the sequential algorithm. We have focused on how to improve the execution of distributed programs by comparing two different load functions. In section 4 we show how an overloaded processor (computer) can affect the global response time of parallel application. Also we show how our deliver function is able to "avoid" overload machines with the information provided by Daem2.

We are, therefore, interested in the design and implementation of a performance management method that includes several important features: ease to programming, a low overhead and accuracy; it must obviously also be a distributed process. Other practical techniques like analytical modeling and simulation
are not considered here. Our method is directed towards accelerating parallel applications in a real distributed system which has a number of collapsed machines.

For each performance study, a set of performance criteria or metrics must be selected [11]. Our Daem2 process works with some of the commonly used performance metrics:

- response time: interval between the initial and final states,
- efficiency: ratio of maximum throughput to nominal capacity,
- utilization: fraction of time the resource (CPU or disk) is busy,
- reliability: probability of errors.

In the design of a distributed-system software monitor, we must take into account the assistance of the services provided by the operating system. General Unix systems provide system calls to get performance metrics such as CPU utilization or memory usage and the number of basic disk operations. Otherwise, the code and overhead of the software monitor can increase and, eventually, become difficult or impossible to implement.

The Daem2 process is a distributed-system monitor that determines the "global load" of different computers working together. Particularly, Daem2 obtains information of disk utilization and CPU utilization from all computers. The process is activated by a timer interrupt with an interval of 10 seconds. When Daem2 has read and computed the CPU and disk utilization, it writes both values in a file in each computer. Daem2 does not introduce overhead since it does not require a lot of CPU time. Our daemon monitor works in milliseconds time resolution and the information provided must be treated as reference.

In this way we can measure CPU utilization when the application to be studied is cpu-bound (has a high rate of computations). On the other hand, if the application is io-bound (has a high rate of input-output operations) the monitor Daem2 can measure disk utilization. Naturally, we should be interested both metrics. Finally, it may be noted that in most computers communication is treated as a low level disk operation (spooling). This is an important issue when PVM or MPI environment is used, where every distributed application has a significant load of communication.

4 Computer experiments

In this section we evaluate the behaviour of our load balancing model. The results were obtained under the following conditions. The application was executed on a cluster of PC 486DX2 at 66 Mhz, with 8 Mb RAM and 1 Gb HD running Linux. The standard MPI implementation from Argonne [8] was used for message passing in our algorithms.

The testing matrices were generated at random. Large size matrices were selected to produce a heavy load so that we might better appreciate the advantages of the technique used. We used rectangular and square matrices.
On the other hand, we have generated the load of the different machines by means of a simple process. This process spends a fraction \( p \) of each observed period performing CPU operations. So that, it is possible to vary the local load in each machine, which allow us to compare our Dynamic function and the uniform function.

### 4.1 Results

In this section, we show some evaluation results. These results were obtained using square matrices of the order of 500 and 1000. We have tested our dynamic function in two stages: Low load stage and heavy load stage. We have executed these two stages iteratively.

The low load stage consists of loading an only node and maintaining no special load on the other computers. This situation can been seen in both figure 1a and figure 2a. The heavy load stage consists of repeating the previous process in a scenario where a set of nodes have a high rate of CPU usage. In this way, we can compare the behaviour of both Dynamic and uniform workload allocation functions in different cases.

![Fig. 1. Response time versus processor utilization. Matrices of the order of 500. (a) Low load, (b) Heavy load](image)

We can see in figure 1 the results of response time for different CPU utilization values when matrices order of 500 are used. In figure 2, it is shown the response time when matrices order of 1000 are employed.

As it can be observed, the Dynamic function gets the best response times whatever is matrix sizes. In proportion as the CPU usage increases, the Dynamic function tends towards the uniform function values. Such reduction of the response time would be significantly higher if the rows amount raises. So that, it is
worth to avoid the overheaded processors because it would give us a considerable save of time and resources. We have measured a time reduction about 60%.

In the case of heavy load it can be noted a bottleneck situation, when a set of machines have reach a high level of CPU usage. This situation can be observed in figures 1b and 2b, particularly in the uniform function.

A special effect may be noted in figures 1a and 2b. The response time of the uniform function increases if the CPU utilization raises. However, the Dynamic function obtains better results for higher utilization. This can be explained if we take into account that the load is low in all machines. A small difference on the load affects to the response time.

5 Conclusions

In this paper, we show the advantages of using an adequate dynamic load balancing model. Our technique is very well suited for parallel applications where a decision about the workload must be made in running time. For this purpose, we have designed a distributed-system software monitor, named Daem2 that gets load information of each node.

In this way we have measured CPU utilization because the application to be studied was cpu-bounded (had a high rate of computations). So, we have shown that it is worth to avoid the overheaded processors because it would give us a considerable save of time and resources. We have measured a time reduction about 60%.

As regards future work we are planning to evaluate the load balancing model for larger networks of workstations and also other parallel algorithms. We are working in the Daem2 process, which will be able to measure other performance metrics.

\[\text{Fig. 2. Response time versus processor utilization. Matrices of the order of 1000. (a) Low load, (b) Heavy load}\]

\[\text{Processor utilization (%)}\]

\[\text{Response time (x 1000 sg)}\]
References


This article was processed using the L\textsc{a}T\textsc{e}X macro package with LLNCS style