SHARING JOBS AMONG INDEPENDENTLY OWNED PROCESSORS

Rafael Alonso
Luis L. Cova

CS-TR-120-87

November 1987
(Revised April 1988)

SHARING JOBS AMONG INDEPENDENTLY OWNED PROCESSORS[†]

Rafael Alonso
Luis L. Cova

Computer Science Department
Princeton University
Princeton, N.J. 08544
(609) 452-3869

ABSTRACT

In a network of *independently owned processors* (e.g., a network of workstations), load balancing schemes cannot consider the whole network as one unit and thus try to optimize the overall performance. Instead, they have to consider the needs of the resource owners. For this type of environment *load sharing* is a more appropriate goal. Load sharing has been accomplished in some systems in an "all or nothing" fashion, i.e., if a node is idle then it becomes a candidate for executing a remote workload, otherwise it is not. This style of sharing is too restrictive in an environment where most resources are underutilized. We present a scheme that replaces this "all or nothing" approach with a gradual one, i.e., where each machine in the network determines the amount of sharing it is willing to do. The scheme, called High-Low, makes sure that the service provided to local jobs of a lightly loaded node does not deteriorate by more than a predefined amount. It simultaneously helps improve the service at heavily loaded nodes. We empirically show that load sharing in a network of workstations can be effective even with a simple-minded allocation of jobs.

April 7, 1988

SHARING JOBS AMONG INDEPENDENTLY OWNED PROCESSORS[†]

Rafael Alonso
Luis L. Cova

Computer Science Department
Princeton University
Princeton, N.J. 08544
(609) 452-3869

1. Introduction

In networks of independently owned processors, load balancing schemes are not appropriate since one cannot consider the whole network as a single unit and thus cannot try to optimize average response time or system throughput. In any load balancing scheme, heavily loaded nodes will obtain all the benefits while lightly loaded machines will suffer poorer response time than in a stand-alone configuration. Users of a frequently loaded machine will cheer for a load balancing scheme while users of a mostly underutilized one will strongly oppose participation in such a scheme. What is desirable is a fair strategy that will improve response time to the former without unduly affecting the latter. The concept of load balancing has to be replaced by the concept of load sharing, i.e., a node will share some of its resources as long as its users are not significantly affected.

In some systems this issue has been resolved by implementing an "all or nothing" strategy. If a machine is completely idle then it becomes a candidate for executing a remote workload. If a machine is being used, even if it is underutilized, then no remote workload is allowed. Basically, any machine can take over an idle one in a master-slave relation, but as soon as the owner of the idle machine uses it (even slightly) all the remote jobs are either put in the background (ran with low priority) [Hagmann1986], moved back to their originating node [Theimer1985],

† This research is supported by New Jersey Governor's Commission Award No. 85-990660-6, and grants from IBM and SRI's Sarnoff Laboratory.

moved to another idle machine [Litzkow1987] or just killed (abnormally terminated) [Nichols1987]. While all of these techniques guarantee the ownership of resources to the owner of an idle machine, they do not assure any performance improvement to the remote jobs the idle node may be servicing. It is more desirable to have a more gradual style of sharing that would attempt to guarantee processor performance to node owners as well as offer some help to the remote jobs that may have been submitted to a node. To do this we introduce the concepts of lending and borrowing resources.

Computers participating in a system where load sharing takes place may be viewed as being at any onetime either a source of jobs or a server of jobs. When viewed as a source of jobs, a machine should only try to execute remote jobs if transferring some of them to another node will greatly improve the performance of the rest of its local jobs. This observation parallels the usual banking practice of borrowing only when necessary. On the other hand, when a computer is viewed as a server of jobs it should only accept remote jobs if its load is such that the added workload of processing these incoming jobs does not significantly affect the service to the local ones. This approach mirrors the sound banking practice of only lending excess funds†. These two notions can be adapted to a load sharing environment via two policies that we denote by **High-mark** and **Low-mark**.

The High-mark policy behaves as follows: each time the execution of a new job is requested at a node the load of the machine is compared against its High-mark value. If the former is greater than the latter then the load sharing mechanism tries to execute the job in a remote host. Otherwise the job is processed locally. Thus, High-mark sets a lower level on the load a machine must have before it begins to transfer jobs to other hosts. Its purpose is to try to reduce processing overhead by load sharing only when the workload of the machine degrades its service dramatically.

† It should be pointed out that the banking analogy does not hold completely. In contrast to the banking situation, borrower nodes need not return their borrowed cycles and lender nodes may not receive back their lent cycles.

The Low-mark policy sets a ceiling on the load a computer may have and still accept incoming remote jobs for service. Its purpose is to be able to handle these incoming jobs while the service to the local ones is not significantly affected. Low-mark works as follows: whenever a request to execute a remote job arrives at a machine, the processor checks if its load is less than its Low-mark value. If so, then the request is accepted and the job is processed locally. Otherwise the request is rejected.

Clearly, High-mark and Low-mark may be implemented together. We refer to this combined policy as **High-Low**. At this point we should contrast this work with that of Eager et al. [Eager1986], who also proposed threshold policies for load sharing. Their analysis used the same threshold value for deciding when to offload work to other nodes (which they call the transfer policy) and for deciding where to run a remote job (which they denote as location policy). In our scheme, the High-mark plays the role of the transfer policy, while the location policy can be chosen depending on the situation. In particular, we use a random allocation of jobs in our experiments. As for Low-mark, it represents a new kind of policy that we call the **acceptance** policy. It reflects the disposition of a node owner to accept a certain level of remote jobs to be serviced by his or her machine. High-mark and Low-mark would be used in addition to location policies to control when a remote interaction should take place. In other words, our scheme is intended to provide the owner's control of a machine independently of the load sharing system.

Finally, throughout this paper, whenever we refer to the load of a machine we mean a consistent load metric that characterizes the usage of that machine. We will be concerned only with the initial placement problem, i.e., where in the network should a job be run, and we will not consider the migration of jobs once they have started execution in a node.

2. The High-Low Scheme

Figure 1 presents the algorithm for the High-Low scheme. For a location policy we chose a random allocation of jobs although it is not a particularly good choice. Under this location policy, the executing node for a job is selected at random and the job is transferred there. No

```
At each node:
    When a local job is invoked:
        IF local_load > High-mark THEN
        BEGIN
            executing_node = location_policy();
            <request execution at executing_node>;
            IF <request accepted> THEN
                <transfer job to executing_node>;
        END
        ELSE
            <execute job locally>;

    When an executing request arrives to a node:
        IF local_load < Low-mark THEN
        BEGIN
            <accept request>;
            <receive remote job>;
            <execute remote job>;
        END
        ELSE
            <reject request>;
```

Figure 1: The High-Low algorithm

exchange of information is done between the machines in the network. It is simple to implement and no system state is available to the nodes. [Eager1986] and [Zhou1987] showed that randomly selecting where to run a job reduces the average response time, but it is very unstable, i.e., its improvements depends on the system workload. As will be seen later, our scheme reduces the instability of the random allocation policy by restricting the interchange of jobs between the nodes.

The salient feature of the High-Low scheme is that two different thresholds are used to decide if a job is to be run remotely. This allows a computer to play multiple roles depending on the values that High-mark and Low-mark take. For example, Figure 2a shows that if the High-mark value is greater than the Low-mark value then the space of possible load values that a machine can have is divided into three regions:

1) **overloaded** (above the High-mark and Low-mark values);

2) **normal** (above the Low-mark value and below the High-mark value);

3) **underloaded** (below both values).

When the load of a machine is in the overloaded region, new local jobs are sent to be run remotely and remote execution requests are rejected. In the normal region, new local jobs run locally and remote execution requests are rejected. In the underloaded region, new local jobs run locally and remote execution requests are accepted.

Most load sharing algorithms use a single threshold (typically the "average" load of all the network processors), and thus only have overloaded and underloaded regions (Figure 2b). High-Low defines a third region (normal) by its use of two different thresholds. This normal region guarantees a predefined level of performance to the node owners. It accounts for the overhead that the load sharing scheme may incur in transferring and receiving a remote job. A job will not be transferred to another node unless it is worthwhile and a remote job will not be accepted unless there is enough excess capacity to handle it.

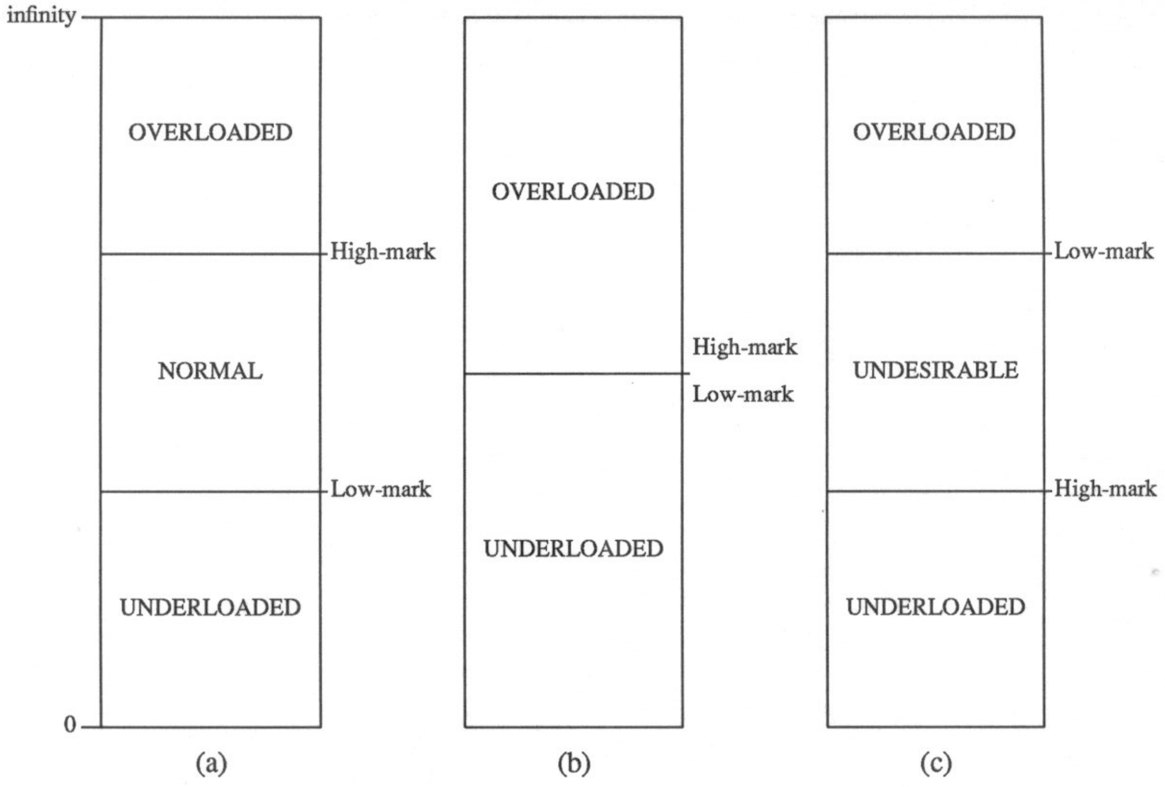

Figure 2: load regions

Notice in Figure 2c that if the Low-mark value is greater than the High-mark value, then a fourth region could be recognized: the **undesirable** (above the High-mark value and below the Low-mark value). In the undesirable region the machine would send its new local jobs to remote processors while accepting remote jobs to be executed locally. Only communication delays are being added to the service time of jobs. To avoid this anomaly, the High-mark value is always greater than or equal to the Low-mark value. However, if High-mark and Low-mark have the same value (Figure 2b) then High-Low will behave as a random load balancing algorithms with threshold.

Figure 3 shows how particular settings of the High-mark and Low-mark parameters correspond to particular modes of operations of a computer. For example, by setting both the

High-mark and the Low-mark to 0 (or the lowest possible value), a computer acts as a job dispatcher (Figure 3a). The computer behaves as if it were always overloaded: it places all of its new local jobs in any available remote machine. This setting could be used to distribute jobs to a pool of processors. If instead, both parameters are set to the maximum possible load value (Figure 3b), then the machine would act as if it were always underloaded, i.e., it would accept any remote process for execution. Thus, the computer is behaving as a process server.

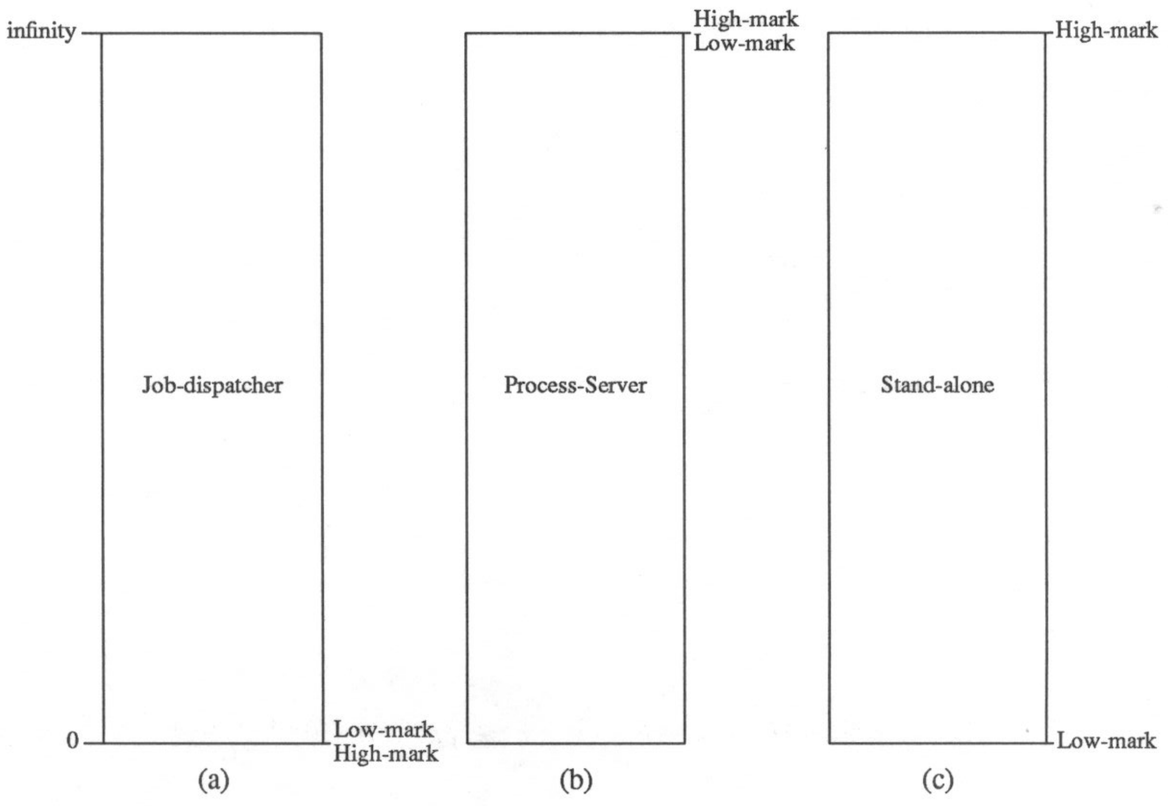

Figure 3: Possible modes of operation

Choosing appropriate values for High-mark and Low-mark is not simple. An automatic fine tuning mechanism together with specifications submitted by machine's owners could be used to obtain the best results. For example, a user could specify that he will allow his machine to process remote jobs if the average response time for his jobs does not deteriorate by more than

10% of the stand-alone time. Selecting the High-mark and Low-mark values is a continuing area of our research. At the present time, we are developing a simple analytical model that uses workload information, user specifications of expected service quality, and machine performance to estimate appropriate values for High-mark and Low-mark.

It could also be possible to vary the High-mark and Low-mark parameters dynamically to allow the computers to have different modes of participation in the load sharing scheme. A system process could set these parameters depending on the number of users, or user processes, in the computer. For example, when the last user signs off, the High-mark and Low-mark values could be set to leave the machine in the process server mode (Figure 3b). As soon as a user signs in, these parameters could be set back to leave the computer in its normal operational mode. If a computer is needed exclusively by its owner then the High-mark and Low-mark parameters could be set to the stand-alone mode (Figure 3c). The attraction of this scheme is that different modes of operation can be easily implemented by dynamically setting the High-mark and Low-mark parameters.

An important observation is that the degree of participation of each computer in this load sharing scheme is completely distributed. It does not depend on any global information or central controller, just on the node's local use and purpose.

Finally, since the load information of a machine represents the available resources in that machine and Low-mark represents the amount of resources that the node's owner is willing to lend, in schemes that require processors to make known their load, the Low-mark value could be included with the machine's load information. In this way other nodes would know in advance the available resources in the network and could plan in advance to use them.

3. Experiments: description and results.

The experimental system we used for our experiments uses a network of workstations. Our environment consist of four identical single-processor machines (SUN 2[†]) connected by an

† SUN 2 is a trademark of Sun Microsystems, Inc.

Ethernet, [Metcalfe1976], and using a fifth machine as a remote file server. The load metric we used for our experiments is the UNIX‡ 4.2 BSD "load average" metric provided by the *uptime* (1) command, [Berkeley1984], and defined as the exponentially smoothed average number of jobs in the run queue over the last 1, 5 and 15 minutes. In our experiments we use the average over the past minute. Our experiments also used numbers related to this "load average" metric as High-mark and Low-mark values.

Using these facilities we implemented the High-Low scheme. We ran several tests with our implementation and compared the obtained results against the situation when there is no load sharing (labeled **alone** in our figures). In our experiments, each user is simulated by a script and the time it takes to complete is calculated (this time is what we define as response time). The script consist of repeated cycles of editing, compiling and running a C program. The C program performs several arithmetic operations.

Emphasis was not only on the average response time of the system (i.e., of all the nodes), but also on the average response time of the jobs at each individual node. This last measurement gives an idea of the changes in local service time when a machine participates in a load sharing scheme.

Figures 4 through 6 show the average response time of the High-Low scheme with a fixed High-mark value (1.75) and several Low-mark values. The High-mark value was selected after running several experiments with just the High-mark parameter [Alonso1987]*. Each figure represents a system with a different workload. Each node in the network has a particular number of users. For example, the distribution "5,1,1,1" represents the number of users in the system, i.e., five users in one computer and a single user in each of the other nodes. This distribution, "5,1,1,1", represents a low system workload, "5,5,1,1" represents a medium one and "5,5,5,1"

‡ UNIX is a trademark of AT&T Bell Laboratories.

* In [Alonso1987], after testing High-mark values ranging from 0.5 up to 3.25, we concluded that using a High-mark value that represents a normal load, i.e., not overloaded and not underloaded, is a good threshold for deciding if a local invoked job should be run locally or remotely.

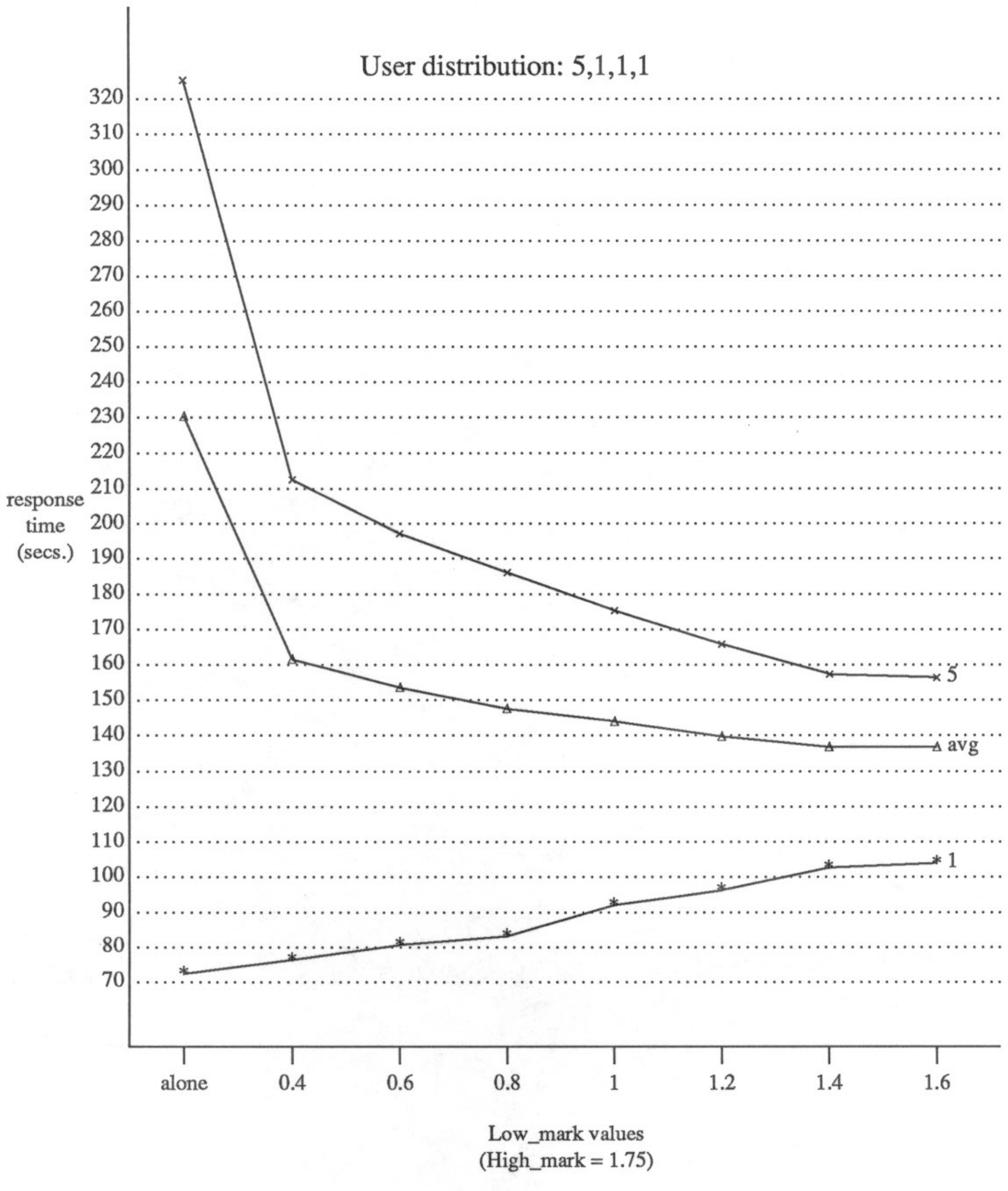

Figure 4: Response time of user jobs using High-Low under a low system load.

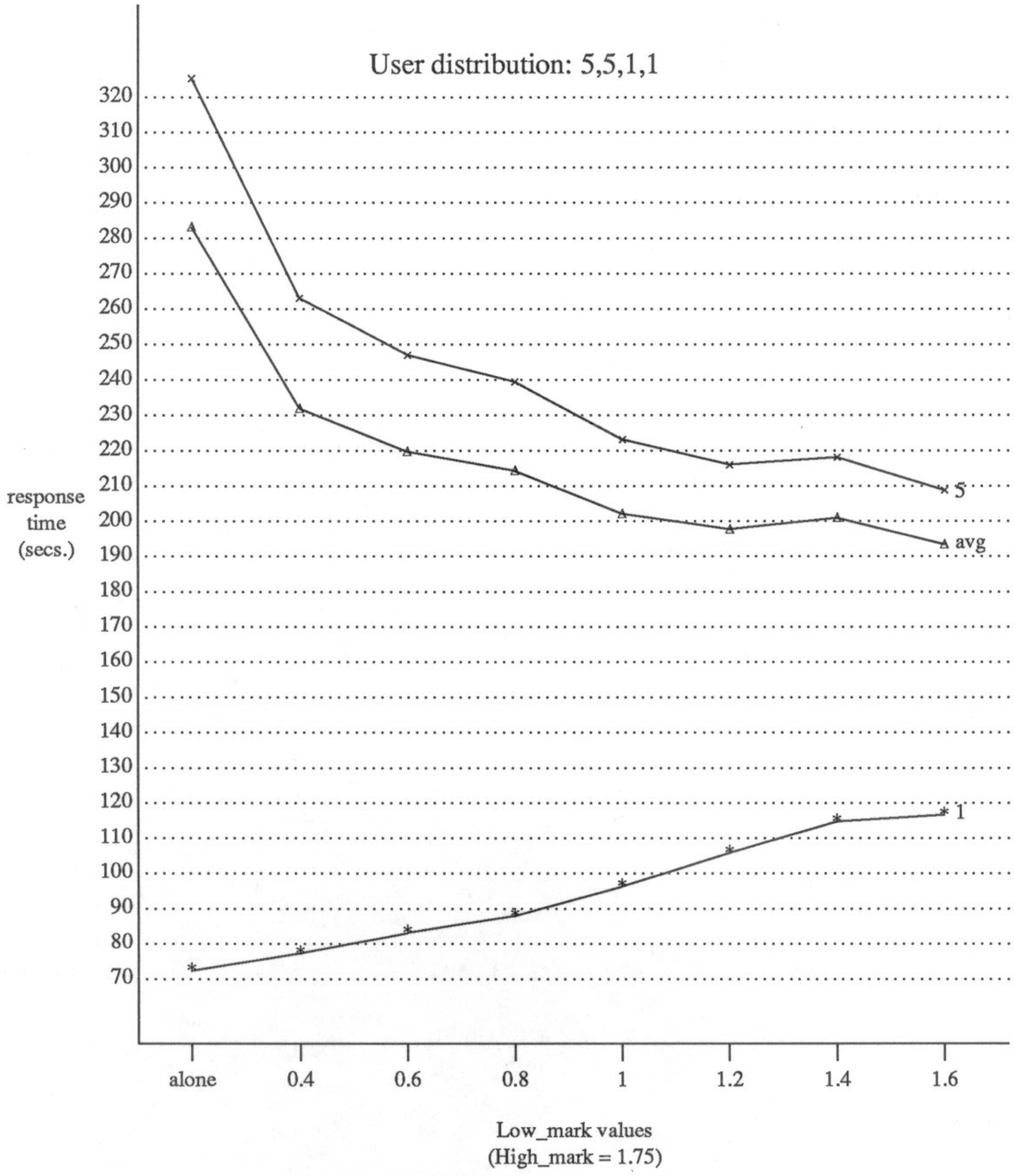

Figure 5: Response time of user jobs using
High-Low under a medium system load.

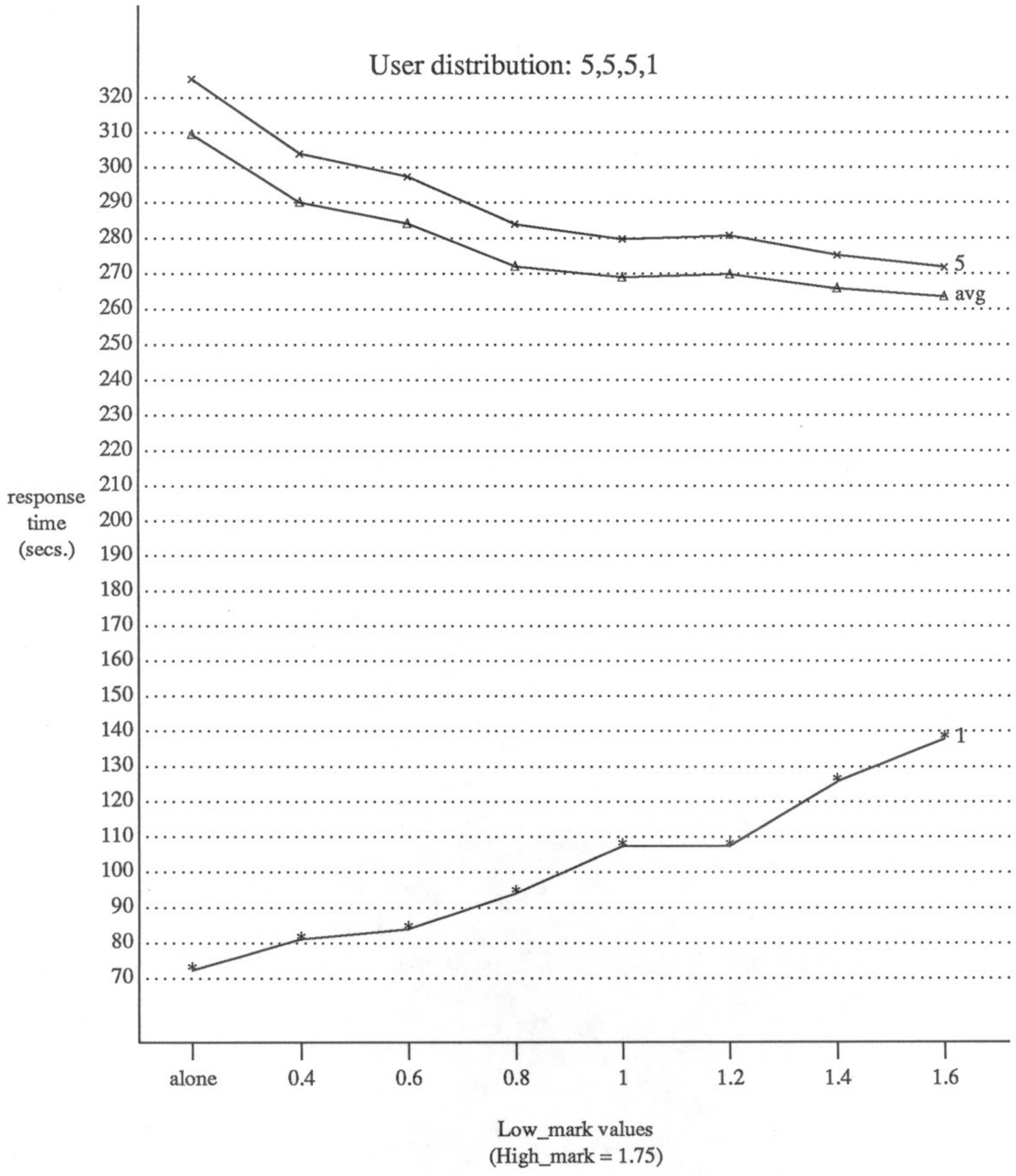

Figure 6: Response time of user jobs using High-Low under a high system load.

represents a high system workload.

The abscissa depicts a range of increasing Low-mark values. These values represents the amount of CPU cycles being dedicated to sharing by each particular node. The labels of the ordinate denote the average response time of jobs submitted by the local users at each node. In each figure there are three curves labeled "5","avg", and "1". The "5" and "1" curves represents the average response time for the jobs submitted by machines with five users and a single user, respectively. The "avg" curve is the average response time of the entire system (i.e., of all the jobs).

Figure 4 shows that even with a small willingness to share (represented by a Low-mark value of 0.4), there is a substantial improvement in the performance of the highly loaded machine while the lightly loaded machines are not significantly penalized. Also, as the Low-mark value increases (more sharing is allowed) the response times of all the machines tend to show a balanced effect. As Low-mark becomes equal to High-mark, High-Low behaves as a random load balancing policy using threshold. Figures 5 and 6 show that even with increased system workload it is still possible to obtain performance improvements at the highly loaded nodes without significantly affecting the service at the lightly loaded ones. Also, notice that in Figures 5 and 6 the average response time (avg.) does not uniformly decrease as the Low-mark increases (Low-mark 1.4 in Figure 5 and Low-mark 1.2 in Figure 6). These "bumps" are produced by the random nature of the job's execution location selection. As it can be seen, even with this anomalous behavior the response time of all the machines is still lower than in the stand-alone case.

Figure 7 presents a summary of our results. In this figure the abscissa represents the system workload, as explained before. For each abscissa label, there are two columns, one for each type of machine in the experimental system (highly loaded ones, with five users, and lightly loaded ones, with a single user). In each column the response time for each Low-mark value is represented. The High-mark value is fixed to the same value as before (1.75). From Figure 7 we can see the incremental changes in performance as we change the Low-mark value.

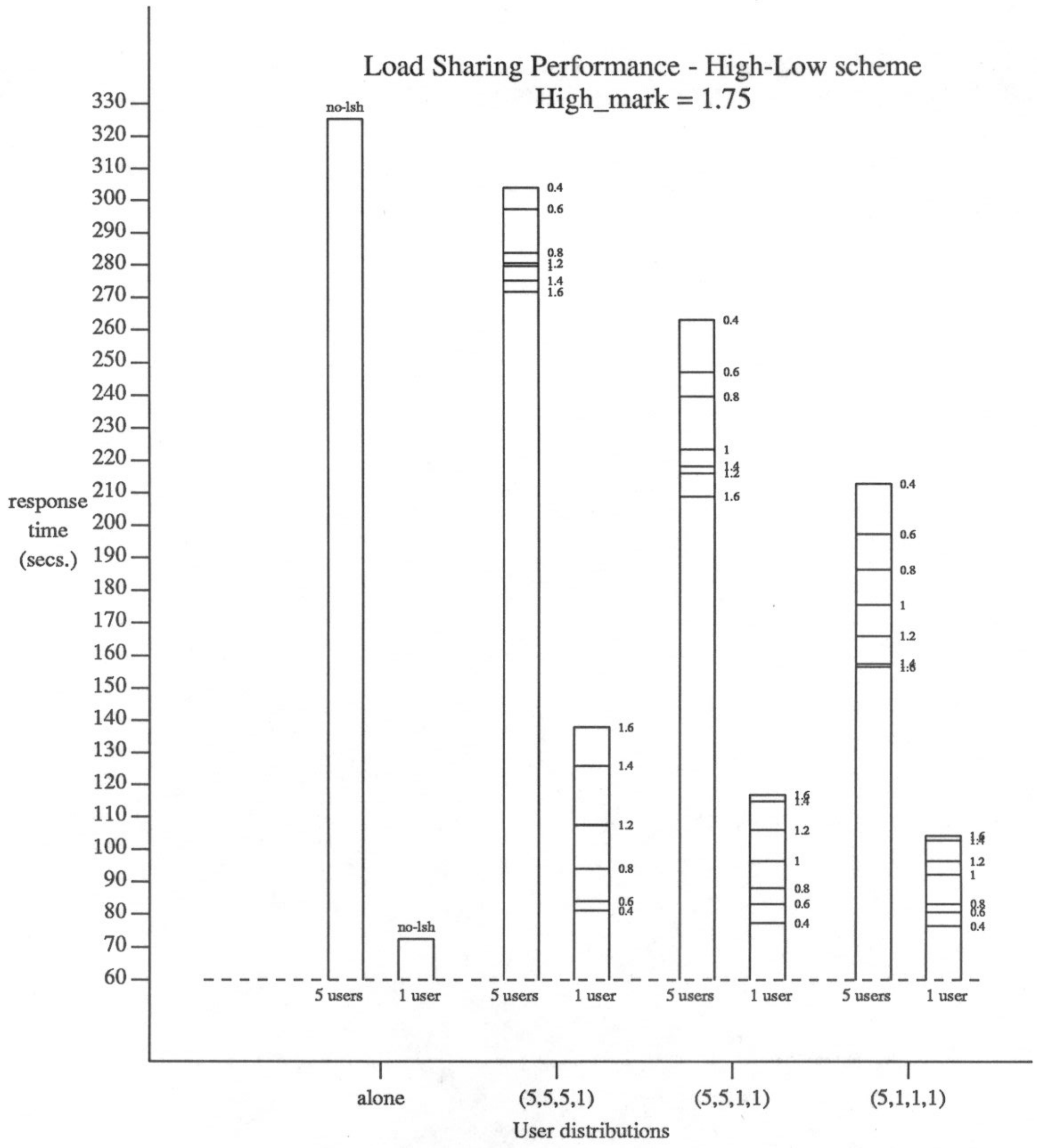

Figure 7: Comparison of different settings of the High-Low scheme under different system loads

It is clear from this Figure that even when there is a lot of activity in the system (user distribution "5,5,5,1"), some improvement in response is achieved by sharing resources in a moderated way. Also, the higher the value of the Low-mark parameter, the more the response time of highly loaded machines improve, but the more the response time of lightly loaded machines degrade.

4. Conclusions

We have discussed the independently owned processors environment, a type of distributed system where sharing the processing capabilities among the nodes improves performance. We have argued that load balancing algorithms are not appropriate for this type of environment because of its characteristics of autonomy and local ownership of resources at each node.

The sharing scheme we have presented, called High-Low, replaces the notion of stealing CPU cycles with the notions of lending and borrowing CPU cycles. Workstation owners do not have to be concern with their resources being abused. The "all or nothing" scheme of load sharing (i.e., only idle nodes are considered for remote load execution) is too restrictive in an environment where most resources are underutilized. Instead, by allowing a moderated sharing among the nodes, good performance is guaranteed to the node owners and to the remote jobs. Also, in a workstation network, having a moderated load sharing allows a lot of distribution cheaply. A natural example is a parallel *make* (1) [Berkeley1984] in UNIX systems.

A further point is that the capability to migrate jobs [Alonso1988] after they have started executing in a machine may be desirable for our system. For example, a process may receive too many remote jobs which can suddenly start to demand a large number of the cycles of the machine, degrading the performance perceived by local jobs. To correct this possible anomaly, the machine could move some of these remote jobs back to their originating nodes or to a new host. In this way control of the local resources could still be maintained.

Finally, we realize that the performance of a workstation does not depend solely on its CPU utilization. We have just used this resource to illustrate our ideas. The notions behind Low-mark and High-mark could also be applied to whatever local resources becomes the system bottleneck, such as physical memory or disk space.

References

Alonso1987.

Alonso, Rafael and Cova, Luis, ''Load Balancing in Two Types of Computational Environments,'' , Princeton University, Department of Computer Science, Technical Report, 1987.

Alonso1988.

Alonso, Rafael and Kyrimis, Kriton, ''A Process Migration Implementation for a UNIX System,'' *Proceedings of the Winter 1988 Usenix Conference*, February 1988.

Berkeley1984.

Computer Science Division, Department of Electrical Engineering and Computer Science, University of California at Berkeley, ''UNIX USER'S MANUAL - Reference Guide,'' *4.2 Berkeley Software Distribution*, March, 1984.

Eager1986.

Eager, Derek L., Lazowska, Edward D., and Zahorjan, John, ''Adaptive Load Sharing in Homogeneous Distributed Systems,'' *IEEE Transactions on Software Engineering*, vol. SE-12, no. 5, pp. 662 - 675, May 1986.

Hagmann1986.

Hagmann, Robert, ''Process Server: Sharing Processing Power in a Workstation Environment,'' *Computing Systems.*, pp. 19-23, IEEE Society, Cambridge, May 1986.

Litzkow1987.

Litzkow, Michael J., ''Remote Unix: Turning Idle Workstations into Cycle Servers,'' *Proceedings of the Summer 1987 Usenix Conference*, pp. 381-384, June 1987.

Metcalfe1976.

Metcalfe, R. M. and Boggs, D. R., ''Ethernet: Distributed Packet Switching for Local Computer Networks,'' *CACM*, vol. 19,7, pp. 395-404, July 1976.

Nichols1987.

Nichols, David A., ''Using Idle Workstations in a Shared Computing Environment,'' *Operating Systems Review*, vol. 21, no. 5, pp. 5-12, ACM Press, November 1987.

Theimer1985.

Theimer, Marvin M., Lantz, Keith A., and Cheriton, David R., ''Preemptable Remote Execution Facilities for the V-System,'' *ACM*, vol. 1, pp. 2-12, 1985.

Zhou1987.

Zhou, Sognian and Ferrari, Domenico, ''An Experimental Study of Load Balancing Performance,'' *Tech. Rept. No. UCB/CSD 87/337*, University of California, Berkeley, January 1987.