LOAD BALANCING IN TWO TYPES OF COMPUTATIONAL ENVIRONMENTS

Luis L. Cova

CS-TR-165-88

January 1988

LOAD BALANCING IN TWO TYPES OF COMPUTATIONAL ENVIRONMENTS †

Luis L. Cova

Department of Computer Science
Princeton University
Princeton, N.J. 08544
(609) 452-5209

ABSTRACT

In many distributed systems it is possible to share the processing capabilities among the nodes. To accomplish this goal, a number of *load balancing* algorithms have been proposed in the literature. The purpose of this type of algorithm is to redistribute the system workload with the objective of equalizing the load at each node. In this report we discuss two types of distributed computing environments for which load balancing techniques yield increased performance: (1) *pool of processors* and (2) *independently owned processors*. Both environments are built around a loosely-coupled computer network, but in the former all computational nodes belong to one user community, while in the latter each machine belongs to a different individual or organization. We argue that these two environments should not be treated with the same type of load balancing algorithm due to their distinct nature. For the first environment we discuss a load balancing algorithm based on executing jobs at the least loaded processor in the network. For the second environment we introduce the concepts of lending and borrowing resources. For both environments we present empirical results to illustrate our approach to load balancing.

January 31, 1988

LOAD BALANCING IN TWO TYPES OF COMPUTATIONAL ENVIRONMENTS †

Luis L. Cova

Department of Computer Science
Princeton University
Princeton, N.J. 08544
(609) 452-5209

1. Introduction

One of the reasons for the existence of distributed systems is to allow resources to be shared across a network of computers in a user transparent way. Work has been done in I/O device sharing, e.g., remote file systems [SUN1986], but one network resource which is seldom shared is the processing capability of the nodes. Because each node usually has its own user community and CPU scheduler, an imbalance of the system workload throughout the network can be a common situation. One solution to this imbalance is to allow users at one node to run processes on other nodes in the network. The usual mechanisms for this are remote logins, e.g. *rlogin* (1) [Berkeley1984], or explicit remote process executions, e.g. *rsh* (1) [Berkeley1984], but for these mechanisms the selection of the execution site and the control of the remote execution is completely up to the users. It is more desirable to dedicate a system program to the task of sharing the processors in much the same way that memory management software allocates the use of memory. Users can then rely on it to automatically handle remote execution of their jobs in order to take advantage of less loaded processors, thus possibly achieving better average response time.

The *load balancing* problem consists of selecting a strategy to redistribute the system workload in a computer network with the objective of equalizing the workload at each node. Recently many load balancing schemes have appeared in the published literature: see [Wang1985] for a proposed taxonomy and a review of the various approaches that have been pursued, or [Zhou1986] for a comparative performance study of several load balancing policies. Load balancing schemes can be

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

divided in two types: static policies, as in [Ni1985a], which ignore the current system state when making decisions and which usually follow average system behavior, and dynamic policies, as in [Eager1986], which rely on system state information. In this report we concern ourselves with dynamic policies.

A load balancing scheme is essentially composed of two parts: a location policy and a load information aspect. The former determines where a job is going to be executed. For example, a possible location policy would run jobs on the machine which has the lowest load. The latter is given by the load metric that is used to determine the load in any given machine. For example, a possible load metric could compute the average number of running processes during a certain period of time. Clearly, the selection of a particular load metric depends in the type of jobs submitted to the machine as well as the node's resources and capabilities. We define the load of a machine as the value of a given load metric for that machine, and the load of a distributed system as a function of the load of all the nodes in the network. In order for a node to be able to make appropriate load balancing decisions it has to gather information about the load of the system. To do this nodes could exchange their loads.

Load balancing algorithms may comprise a variety of strategies to exchange load information between the nodes of a network. These strategies can range from not exchanging any information to having the nodes possess a common knowledge of the system's load. The former extreme could be implemented by choosing execution sites by means of educated guesses, and the opposite extreme could be implemented by shared memory, a central controller, or by exchange of frequent messages. There is a tradeoff between these two extreme cases: inaccuracy of the decision made vs. overhead in the decision process. In between these two situations there are many schemes to be explored. The selection of one over another for a particular distributed system is a design issue that affects the service to its users.

Design issues, like the one just mentioned, are influenced by the configuration that the distributed system has. The type of computational environment in which the load balancing strategy is to be implemented has a large impact on the design of the load balancing mechanism. The overall objective of the distributed system has to be taken into account when choosing a strategy because it influences the amount of information to be exchanged among the machines as well as the type of location policy to be used. This relation has been given very little attention by the implementors of load balancing mechanisms.

Distributed systems have some typical configurations. For example, a *pool of processors* is a group of interconnected computers dedicated to servicing equally a group of users. The goal is to provide the user community with improved service. Most of the work described in the load balancing literature has dealt (explicitly or implicitly) with this type of environment ([Stankovic1984], [Alonso1986a], [Eager1986], [Zhou1986]). In this environment all the nodes in the network functionally belong to one organization. The appropriate load balancing scheme for this environment should focus on enhancing the overall system performance which is the system's goal. Examples of this type of system are most of the computer centers in industries and universities.

Another type of distributed system is the *independently owned processors* environment. In this environment each node of the network functionally belongs to a different user, whether that is a single user or a group of users. Instances of this environment are networks of workstations, and networks of inter-departmental machines. Although a load balancing scheme can be used in this type of system, it cannot be treated with the same techniques used with a pool of processors. For this type of environment, *load sharing* is more appropriate. Load sharing entails a redistribution of the system workload, but not necessarily resulting in an even distribution of work.

In the following sections we expand the description of the two environments listed above by focusing on why different schemes for load balancing are needed to satisfy their objectives. In Section 2 we describe further the pool of processors model and we show empirical results of a scheme which enhances global system performance. In Section 3 we discuss the independently owned processors environment and the concepts of *lending* and *borrowing* CPU cycles. In Section 4 we suggest a scheme which can be used in mixed environments and, finally, in Section 5 we present our conclusions and ideas for future research.

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 a job should be run, and we will not consider the migration of jobs once they have started running in a node.

2. Assigning jobs to a pool of processors

A pool of processors is a group of interconnected computers dedicated to equally serve a group of users. The goal is to provide the overall user community with improved services which might yield lower response time, higher throughput or greater fault-tolerance.

As mentioned before, most of the work described in the load balancing literature deals (explicitly or implicitly) with the pool of processors environment. For example, Chow and Kohler [Chow1979] presented the results of a queuing modeling study where the goal is either to reduce the average job response time, or maximize the system throughput, or minimize the system time. The models used are based on a central dispatcher doing the job assignment to the different processors. Their results showed that models incorporating load balancing schemes in the job scheduling algorithm have a lower job turnaround time than when load balancing schemes are not used. Many other studies from different authors followed this one all focusing in reducing average response time as their main goal.

In [Stankovic1984], the simulation of three dynamic schemes is presented. All of them select the least loaded machine as the executing node for an arriving job. In the first algorithm, a job is transferred by a node if the difference between the load of that machine and that of the least loaded node is greater than a fixed bias. In the second algorithm two biases are used. If the difference (the one stated before) is greater than the first bias, but less that the second bias then one job is transferred. If the difference is greater than both biases then two jobs are moved. The third algorithm is similar to the first with the addition that after a job is transferred to a machine the node that is doing the transfer updates its local information about the machine where the job has been sent, and does not transfer jobs to the same machine for a fixed period of time.

Krueger and Finkel [Krueger1984] presented a preemptive load balancing algorithm called "Above-Average." The scheme migrates processes from machines whose load are above the system average to machines whose load is below such average. The idea is to reduce the variance in load among the nodes in the network. The average load of the system is maintained dynamically by allowing any node to broadcast an updated system average whenever it believes that the current value of the average load is inaccurate. A node assumes that the current system load average is inaccurate whenever its own load is very different from the average and there is no other node trying to interact with it.

In [Ni1985b] a load balancing scheme called the "Drafting" algorithm is presented. This is a type of bidding algorithm in which lightly loaded nodes send messages to heavily loaded nodes requesting jobs to be run. Loaded nodes then send back their jobs information and the lightly loaded nodes decide which job to run.

Eager et al. [Eager1986] discussed the issue of the appropriate level of complexity for load balancing policies. Simulation results for three schemes of increasing complexity are compared. In their simplest scheme the destination node (where the job is going to be run) is selected at random. In their next scheme in complexity, a destination node is randomly selected and it is probed to determine whether the transfer of a job would place its load over a predefined maximum value. If this is true, no job is moved to the selected node and a new node is chosen and again probed. This process continues until either the number of probes exceeds certain limit or a suitable node is found. Their most complex scheme involves choosing several distinct nodes at random and polling each of these nodes to determine the least loaded one. Once the least loaded is chosen, a job is transferred to it if the added load does not increase its load over a limit, as in the previous scheme.

In [Huang1986], three dynamic probabilistic load balancing algorithms are described. In all three algorithms, each node in the network computes an average estimate of the load of the whole system from load information broadcast by each node. In their first scheme, the load information received by each node is used to compute an estimate of the average amount of unfinished work in the network. Once a job arrives at a node, the estimate of unfinished work in that node is compared to the computed average. If the former is greater than the latter by more than a fixed constant, then a remote node is picked at random and the job is transferred there. In their second algorithm, each node associates a probability value to each of the other nodes in the network. This probability is computed from the relation between the local estimate of unfinished work and the estimate of unfinished work received from the other nodes. If a job arrives at a node and it decides to run the job remotely then a site is selected using the computed probabilities. Their last scheme is similar to the previous one, but instead of estimating unfinished work they use estimated job queue length at each node, which is simpler to compute.

In [Zhou1986] a trace-driven simulation study of seven different dynamic load balancing schemes is presented. The seven algorithms were chosen because they

represent a large family of schemes: server vs. source initiated interaction, periodic vs. on-demand load information exchange, and system-wide, subset or random selection of execution site. This work together with [Zhou1987], an empirical study of five out of the seven schemes proposed in their previous report, is one of the most complete studies in dynamic load balancing done up to date.

We can see several similarities among these previously described schemes. Most of them are decentralized in control and each node in the network frequently acquires information about the local state of the other nodes. This information is used to select remote execution sites. An implementation of a load balancing scheme done in the Distributed Computing Laboratory at Princeton University's Department of Computer Science has many of the attributes described above. We will focus our discussion on this implementation and through it we will show the applicability of such load balancing schemes to pool of processors.

2.1. Lsh: a "least loaded" load balancing scheme.

In [Alonso1986a] a load balancing scheme (called **lsh**) was developed based on broadcasting local system state to all the nodes in a local-area network (LAN) and on transferring jobs to the least loaded node. Each node reaches load balancing decisions in a decentralized fashion, i.e., without the existence of a central controller. The purpose of the prototype was to demonstrate that sizable overall system performance gains could be achieved using a simple load balancing mechanism on top of an existing system with small overhead and making very few changes in the underlying software. It was noticed that having accurate information about the entire system was expensive because processing broadcast messages from other nodes takes a substantial amount of CPU cycles. There is a tradeoff between the broadcasting interval and the processing overhead which directly affects the accuracy of the information on which a machine has to base its locality decision. In a follow-up study [Alonso1986b], this tradeoff was discussed and the issues involved in evaluating load metrics and decision policies were described.

Lsh was revised and improved to take care of obvious flaws that a simple "least loaded" scheme has. These problems are the swamping and drought effects. These effects are produced by the same factor: outdated system state information due to update interval and communication delays. In [Williams1983] the importance of this factor is recognized in the design of load balancing strategies.

In the swamping effect many jobs are sent to one machine (the least loaded at that moment) before it can broadcast its new load. This occurs because several

machines may choose to transfer jobs to the least loaded site within the same small interval of time, before new state information from the least loaded machine is broadcast. Therefore, the response time of these transferred jobs may be even greater than if they had been processed in their originating nodes.

In the drought effect, truly least loaded nodes do not receive remote jobs. This happens because the moment a machine gets less loaded or even idle is not synchronized with its broadcast interval. A node may be the least loaded site in the network, but until it broadcasts its new state, no other node will know it.

To correct the above described anomalies two policies were incorporated to lsh: required load difference and implied load. The required load difference limits a machine to send one of its jobs to a remote node only if the difference between its load and the load of the remote machine is more than some specific amount. This would reduce remote execution overhead. With implied load the system load information kept at a machine is updated each time a local job is migrated to another node, i.e., when a machine transfers a job to another node, it adds a certain amount to its information about the receiving node. This is done to compensate for the added workload at the remote site, until an updated load message is received. In this way the sending machine has more accurate information when making the next load balancing decision.

So far we have presented different details which have to be taken into account when designing or choosing a load balancing scheme for a pool of processors. Eager et al. [Eager1986] suggested that " ... *extremely simple load sharing policies using small amounts of information perform quite well - dramatically better than when no load sharing is performed, and nearly as well as more complex policies that utilize more information* ... " . Although we agree with most of this statement we believe that more complex policies are more stable and robust even though their gain in performance is small compared to simpler schemes.

To explore this possibility we have implemented a random job allocation policy (labeled **random** in our figures). This policy selects an executing node at random and the job is transferred there. No exchange of information is done between the nodes. We have compared this policy against the naive implementation of lsh (labeled **lsh** in our figures) and the best choice of parameters for the current version of lsh (labeled **lsh-best** in our figures).

The system we used for our experiments uses a network of workstations. Our environment consists of four identical single-processor machines (SUN 2†)

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

connected by an 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 last minute. This is the load metric we used for all the experiments in this report.

2.2. Performance of lsh.

In figure 1 we show the average response time of all the schemes under different user distributions. The abscissa depicts the number of users per machine, which gives a sense of the balance of the system. The further to the right along this axis the more unbalanced the system load is. The labels in the ordinate denote the values for the average response time (in seconds) of jobs submitted to the system. Each user is simulated by a script and the time the script takes to complete is defined as the response time of each user job. The script consists of repeated cycles of editing, compiling and running a C program. The program performs several arithmetic operations.

As can be seen, all the policies behave well under unbalanced distributions, but under balanced ones, lsh and random do worst than the "no load balancing" case (labeled **no-lsh** in our figures). The random policy spreads jobs around the system equalizing the job queues among all the nodes. Under balanced distributions, this behavior represents worthless job transfers which add significant communication delay to the jobs' response time. Notice that all the policies tend to smooth the value of the average response time for any distribution of users. This is a desired behavior because users can expect their jobs to complete within a certain range of time, regardless of the load distribution in the system.

In figure 2 the average standard deviation for all the policies is shown. Note that the value range of lsh-best's standard deviation is the smallest of all the policies. This suggests that the most complex policy has a better stability than the simpler policies.

---

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

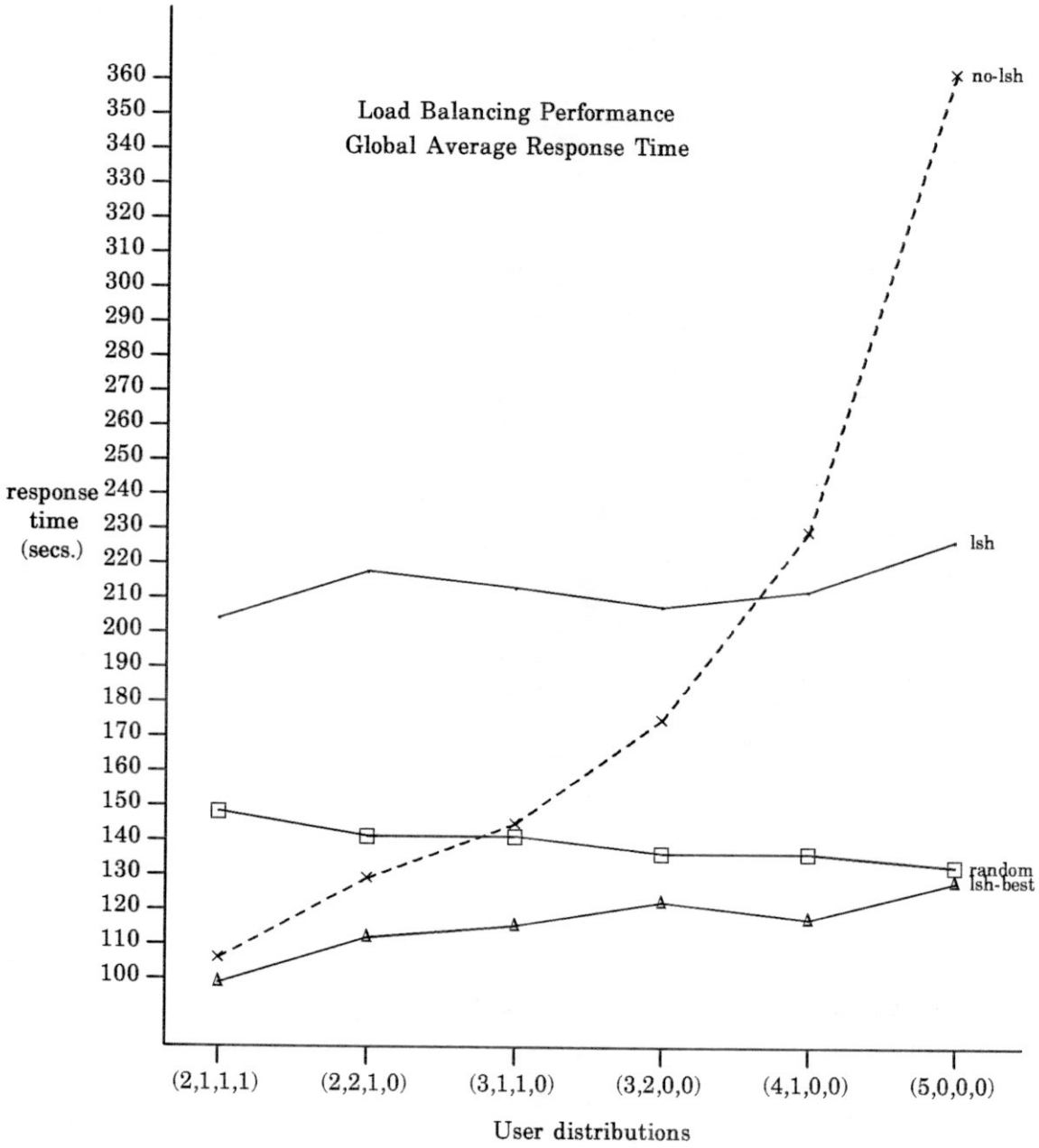

Figure 1: Comparison of different load balancing policies

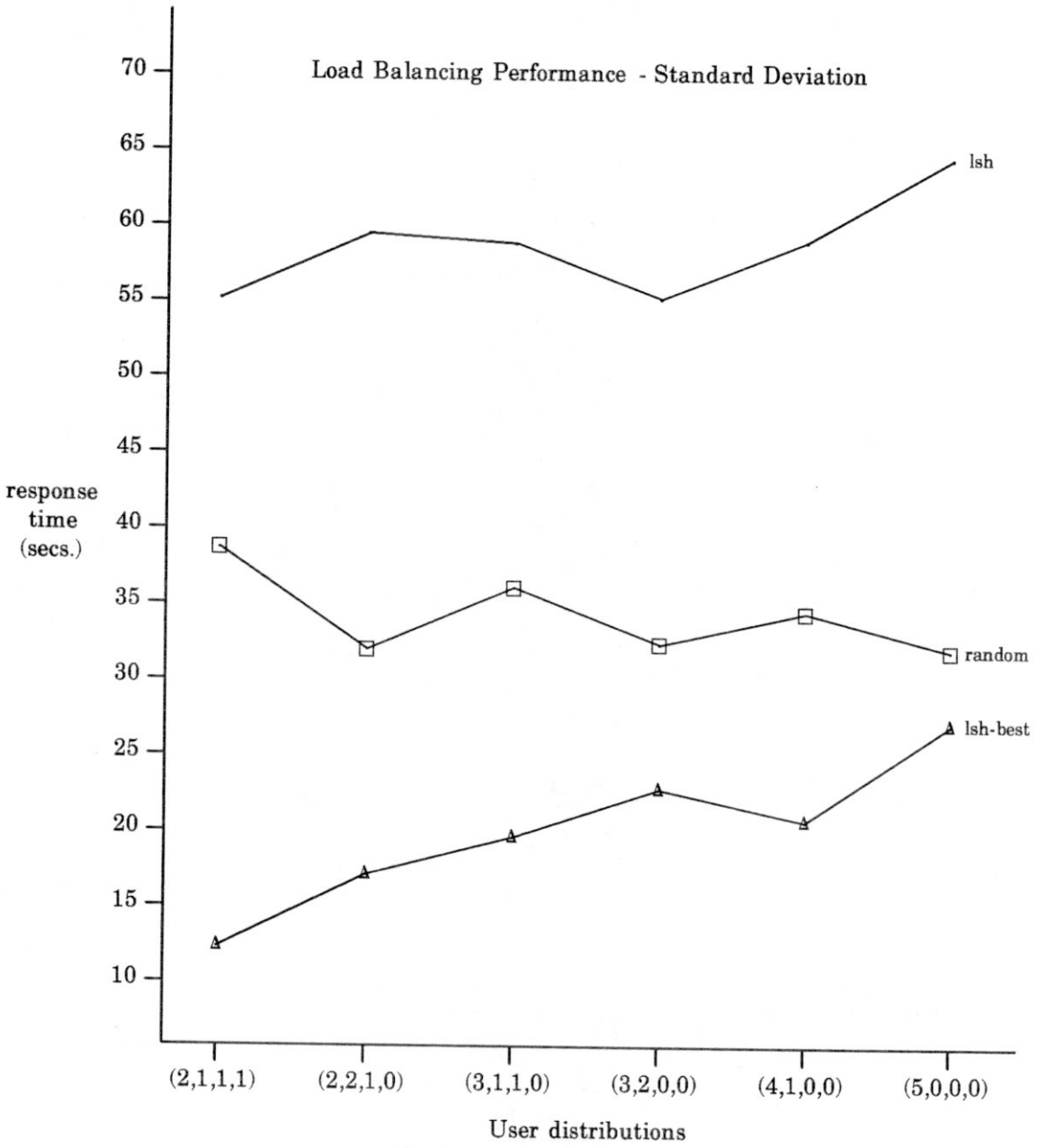

Figure 2: Comparison of the standard deviation of different load balancing policies

3. Sharing jobs among independently owned processors

Computer networks were initially intended to allow communication between independent machines owned by distinct organizations. At first, most of the communication was done through point-to-point connections (long-haul networks), but since the advent of broadcast technology, local distributed computer systems have evolved around it (local area networks).

Many researchers have studied the CPU sharing issue in relation to the independently owned processors environment. For example, Theimer et al. [Theimer1985] proposed to use idle nodes as computational servers in networks of workstations. It is clearly noted in this work that using idle workstations to serve remote jobs should not conflict with the use of the machines by their owners , i.e., owners should not see a higher response time for their jobs. Theimer *et al.* meet this requirement by using priority scheduling. Locally invoked jobs to a workstation have a greater scheduling priority than remote jobs. While this priority scheme assures the availability of resources to the owner of a workstation, it does not guarantee any performance improvement to remote jobs.

Hagmann [Hagmann1986] presented an experimental system for distributing computations normally performed on a workstation to some collection of machines, possibly other workstations. In the current state of the system, a machine can simultaneously be serving local jobs and remote jobs. An owner of a workstation that is serving remote jobs will only perceive an increase in the machine's cpu utilization. Ownership of resources is guaranteed in the same manner as in the previously described system.

Zhou [Zhou1986] studied the effect of load balancing schemes on individual nodes. His simulation results indicated that the performance of all nodes, even the lightly loaded ones, is generally improved. However, in a follow up study Zhou and Ferrari [Zhou1987] empirically showed that the reduction in local response time does not occur on machines that were originally lightly loaded. This effect is undesirable with independently owned processors. Clearly, it is important to develop a technique that minimizes the performance degradation of the lightly loaded nodes.

On one hand, users want faster response time which could imply greater load distribution. On the other hand, the owner of a particular node will only participate in the load balancing scheme if the benefits he will receive are far more than the cost he will have to pay. The cost is usually reflected as the amount of CPU time dedicated to service jobs from other nodes.

With any load balancing scheme, heavily loaded nodes will get all the benefits while lightly loaded machines will have poorer response time than in a stand-alone fashion. Users of a frequently heavily loaded machine will cheer for a load balancing scheme while users of a frequently lightly loaded one will strongly oppose participation in such a scheme. What would be desirable is a fair strategy that will improve response time to the former without significantly affecting the latter. With independently owned processors , load balancing schemes cannot consider the whole network as one unit and thus try to optimize average response time or system throughput. Instead they have to consider the main objective of the environment: " *guaranteed delivery of resources to owners* . " The sharing of resources should behave according to the same rules of borrowing and lending that a commercial bank may have, i.e., a machine should only transfer jobs to another node if it is really necessary (borrowing CPU cycles) and it should process incoming remote jobs if by doing so the service to the local user is not significantly affected (lending CPU cycles). We will now focus on these two ideas.

3.1. High-mark and Low-mark schemes

Computers participating in a system where load balancing takes place may be viewed as being either sources of jobs or servers of jobs. When a machine is viewed as a source of jobs it should only try to remotely execute jobs if by doing so the performance of the rest of its local jobs is greatly improved. It is only then that the load balancing mechanism should automatically transfer jobs. It emulates the principle 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. It emulates the principle of lending only when affordable. These two notions can be adapted to a load balancing environment via two policies: **High-mark** and **Low-mark**.

The High-mark policy behaves as follows: each time a new job is invoked 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 balancing mechanism tries to execute the job in a remote node. 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 nodes. Its purpose is to try to reduce processing overhead by load balancing only when the workload of the machine degrades its service dramatically.

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 guarantee that the machine will 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.

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 adequate values. 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. As it will be seen later, our experiments suggest that appropriate values depend on the load distribution in the system.

As we have mentioned before, this type of environment fits the description of most long-haul networks as well as the description for the more recent networks of workstations. Since the communication delays of long-haul networks make load balancing impractical, we will use the latter type of network to further describe the independently owned processors environment.

3.2. An Example: The Network of Workstations.

The fast development of computer technology has made more powerful and smaller processors and peripherals available at smaller prices. Because of this, powerful personal computers (workstations) can be bought and used by a single user. These workstations have processors of about 1-3 MIPS, main memory of about 0.5-2 MBytes and are usually used with magnetic disks of about 50-120 MBytes or with specialized computers acting as file servers. The basic purpose of these workstations is to insure the availability of resources to their owners, but many resources or peripherals are still either too expensive to be owned by a single user (e.g., magnetic disk pack drivers) or are rarely used thus making it impractical to assign them to one person (e.g., phototypesetters). Networking is then a way of sharing this type of devices between workstations.

Workstations have been usually connected to local area networks using broadcast media (e.g., Ethernets) or by a token ring [Farber1972] . Many research laboratories as well as universities have assigned workstations instead of simple terminals to their members. The usefulness of this type of network grew with the inclusion of remote procedure calls, shared or global file systems and network operating systems. This collection of workstations became a rich and powerful computing environment. However, often a user cannot take full advantage of the available resources. For example, in many cases it is burdensome for an unsophisticated user to simultaneously use more than one machine to service his jobs. Thus load balancing techniques can be used to achieve higher availability of processing power in a user-transparent way.

The Distributed Computing Laboratory at our department is an example of the workstation network just described. Using the laboratory's facilities we implemented two versions of the random policy to test the two concepts explained before: High-mark and Low-mark. One implementation incorporates High-mark and the other incorporates Low-mark. We have run several tests and compared the results obtained against the pure random scheme and lsh-best. For these tests we followed the same methodology described in section 2. However, emphasis was not only on the average response time of the system, i.e., of all the nodes, but also in the average response time of the jobs at each individual machine. This last measurement gives an idea of the changes in local service time when a machine participates in a load balancing scheme.

3.3. The High-mark scheme.

Figures 3 to 5 are related to the High-mark scheme. Our implementation uses numbers related to the "load average" calculated by the UNIX 4.2 BSD operating system as High-mark values. The High-mark scheme is implemented as follows: whenever a new job arrives at a machine, its load value for the last minute is compared to the High-mark value. If the former is greater or equal than the latter then a remote machine is randomly chosen and the job is transferred there for execution.

One set of our experiments tested High-mark values ranging from 0.5 to 3.25 (increasing 0.5 each time). For our discussion we chose three values which represent three broad classes: low, medium and high High-mark values. In figure 3 we show a comparison of the High-mark scheme with the three significant values: 0.75, 1.75 and 3.0. Notice that a small value for the High-mark parameter

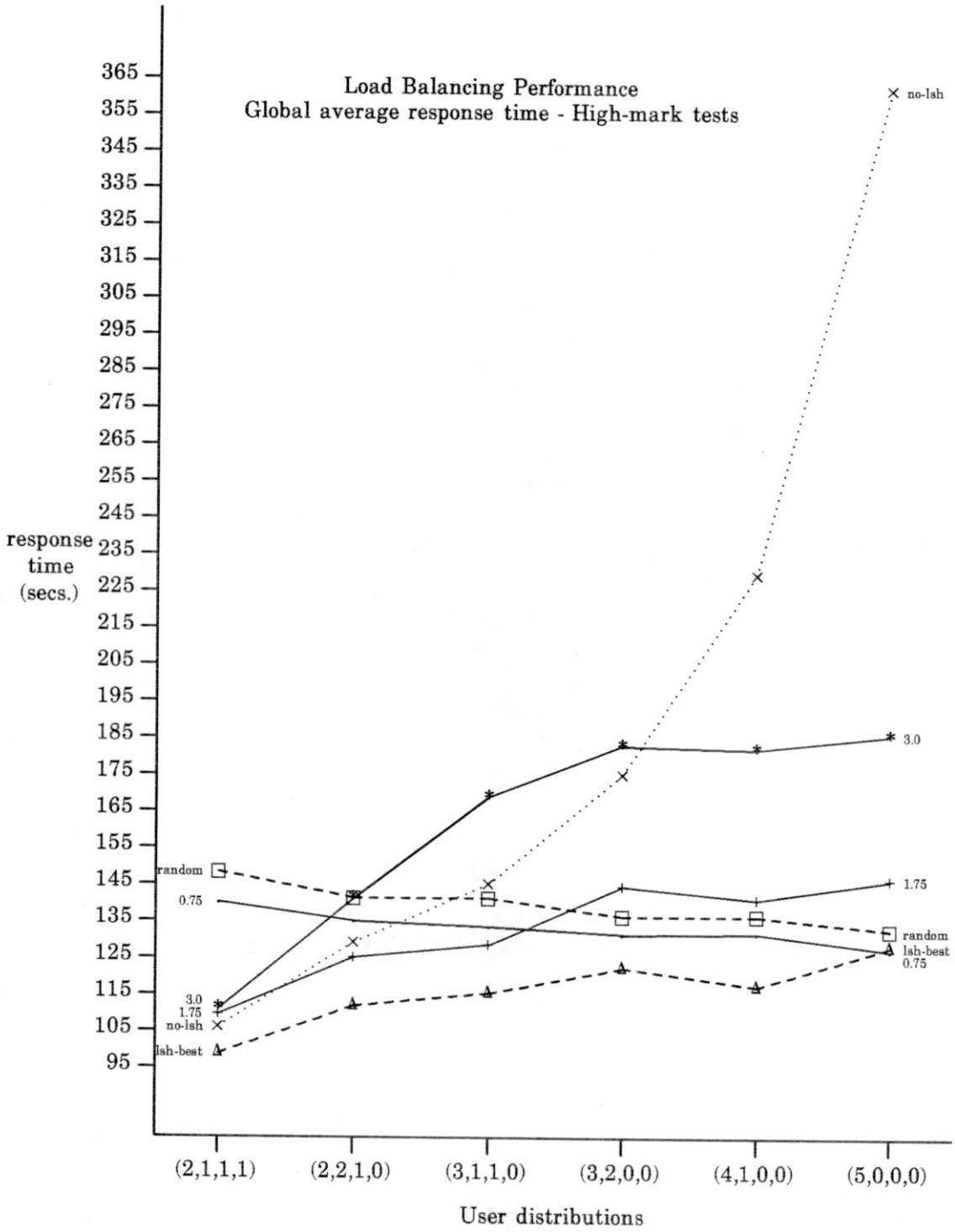

Figure 3: Response time of user jobs using the High-mark scheme

allows the utilization of more remote processors. For unbalanced distributions of users this would mean lower average response time for the system. In balanced cases the transfer of a job would only add communication delay to its service time, thus the average response time of the system would not improve. For a higher High-mark value the situation is the opposite. Most of the work is done by the local processor. This situation is desired when the system is balanced because each machine will spend most of its time processing local jobs and will not waste CPU time in handling remote jobs. Communication delay will not be added to the response time of jobs. Dynamically setting the High-mark parameter depending on the system's load distribution can be a possible option for a load balancing scheme. Meanwhile, using a High-mark value near the median will provide an appropriate starting point for our experiments.

In figures 4 and 5 we are comparing local average response time of the least loaded machines (1 user per node) and that of the most loaded machines under several load balancing schemes. In these Figures the abscissa represents the system workload, as explained before. For each abscissa label there is a column in which the response time for each tested scheme is marked. The point we want to show is that in the independently owned processors environment it is important to use strategies in which the local average response time of a machine does not get much worse compared to its stand-alone response time.

In figure 4 we observe that the High-mark scheme with value 3.0 keeps the local average response time of the single user machines lower than in the no-lsh or lsh case. This is because idle workstations are being used. Idle workstations can process remote jobs very fast and the non-idle nodes are freed to a level where job processing is done quickly. Clearly, load balancing schemes should take advantage of idle sites first. Looking at the same scheme in figure 5 we see that the situation is not the same for the most loaded nodes. In some load distributions the local average time is better, but not significantly. If we now pay attention in the same two figures to the High-mark scheme with value 0.75 we note that we draw opposite conclusions to the ones drawn for the value 3.0. Finally, looking at the High-mark scheme with value 1.75 and lsh in both figures, we note that the local average time is also improved for the least loaded machines as well as for the most loaded machines in almost all the distributions. This is the type of effect we are looking for, where global average response time is improved and single user machines do not get significantly penalized for collaborating in the load balancing scheme.

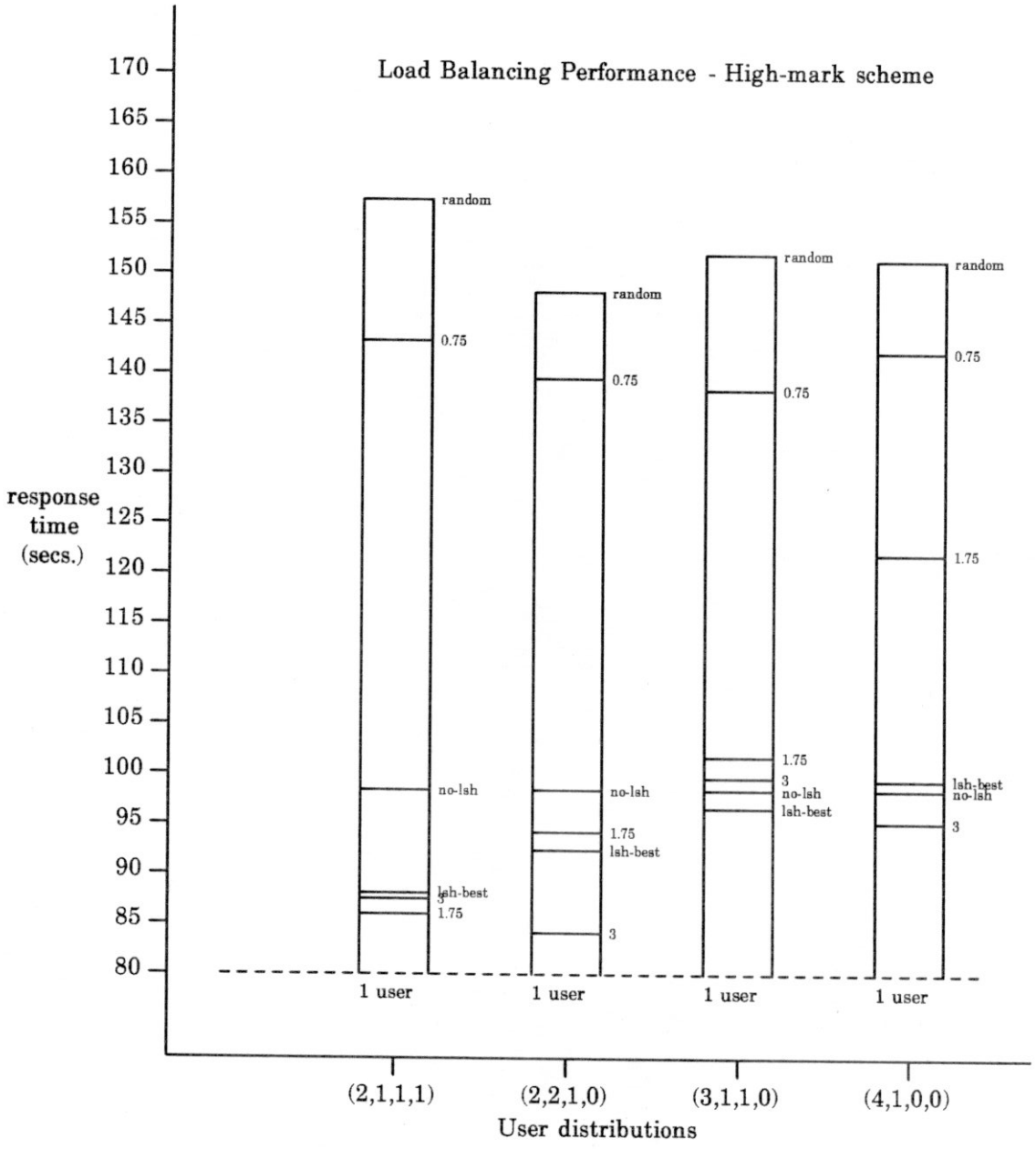

Figure 4: Local response time for
the least loaded nodes using High-mark

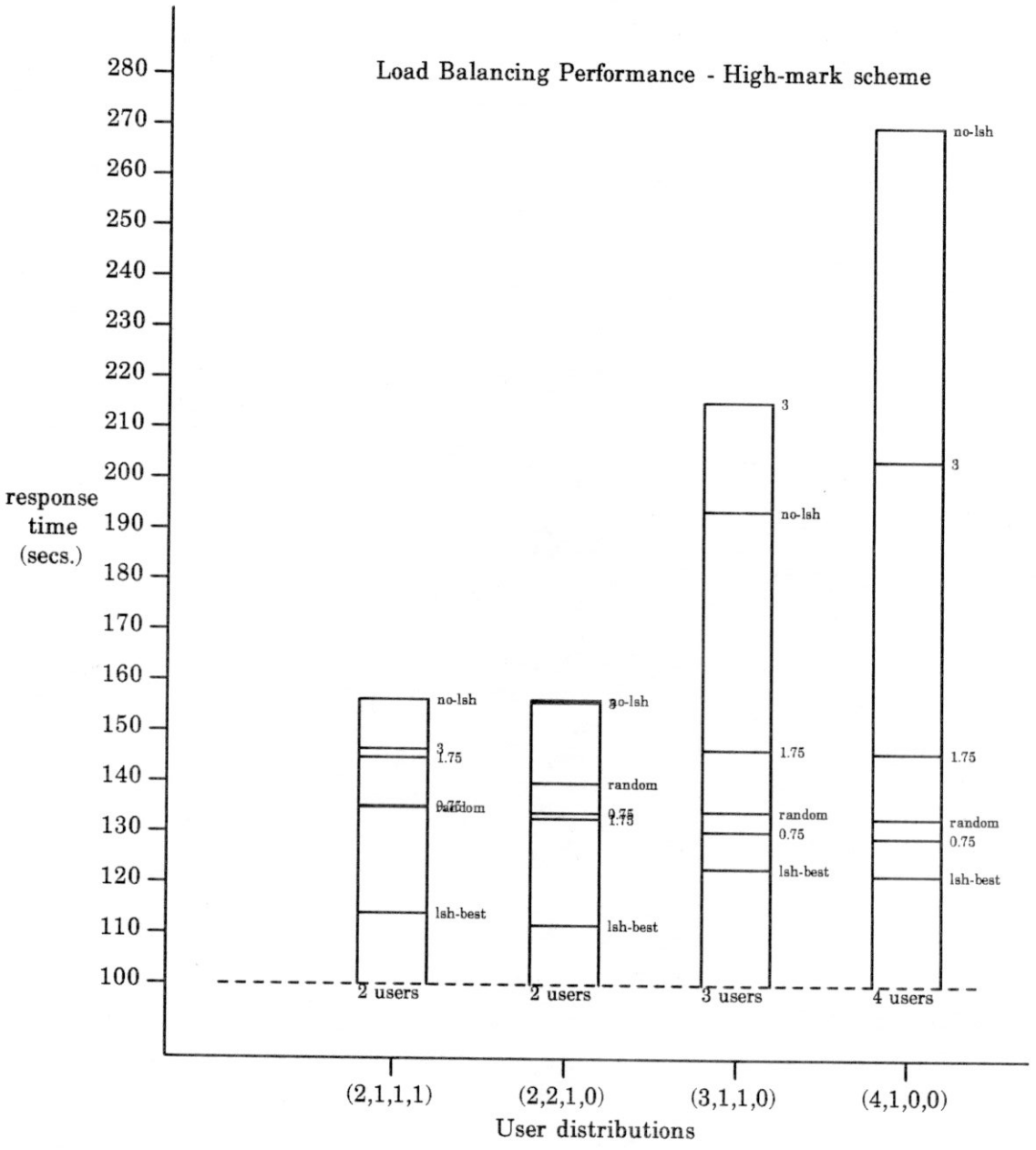

Figure 5: Local response time for
the most loaded nodes using High-mark

3.4. The Low-mark scheme.

Figures 6 to 8 are related to the Low-mark scheme. As with the High-mark scheme, we used UNIX "load average" numbers to characterize the Low-mark values. The Low-mark scheme is implemented as follows: whenever a request for a remote job execution arrives at a machine, its load value for the last minute is compared to the Low-mark value. If the former is less than the latter then the request is accepted and the remote job is executed. In any other case the request is rejected and the job is executed in its originating node.

We followed the same methodology used for the High-mark scheme and explained in the previous subsection. We tested different Low-mark values ranging from 0.5 to 3.25 (increasing 0.5 each time) and we chose three values: 0.5, 1.5, 2.25 to represent low, medium and high Low-mark values. In figure 6 we show a comparison of the Low-mark schemes with the chosen values. It can be noted that a low value allows little remote execution, situation that is desirable under balanced load distributions, but not so under unbalanced ones. On the other hand, as the Low-mark value increases, more remote executions are allowed thus making it more appropriate for unbalanced distributions, but not for balanced ones because only communication delay time would be added to the jobs's service time. Dynamically setting the Low-mark value depending on the system's load distribution can be a possible option for a load balancing scheme. Meanwhile, using a value near the median will provide an appropriate value for most of our experiments. Note that the conclusions drawn so far about the system's load distribution and the Low-mark value are opposite to the ones drawn for the High-mark value. The same will be true for the next two figures.

In figures 7 and 8 we compare local average response time of the least loaded machines (1 user per node) and that of the most loaded machines. In figure 7, we can see that the 0.5 value achieves lower response time than the no-lsh case. This occurs because single user machines are exclusively dedicated to their local jobs, rejecting most of the remote execution requests. Instead, in figure 8, we notice that this Low-mark value is not appropriate for the most loaded machines under unbalanced distributions. Paying attention to the high value in the same two figures, we get opposite conclusions as those drawn for the low value. Finally, looking at the scheme with value 1.5 we get the effect we are looking for: global average response time improves and the response time of the least loaded machine does not deteriorate significantly.

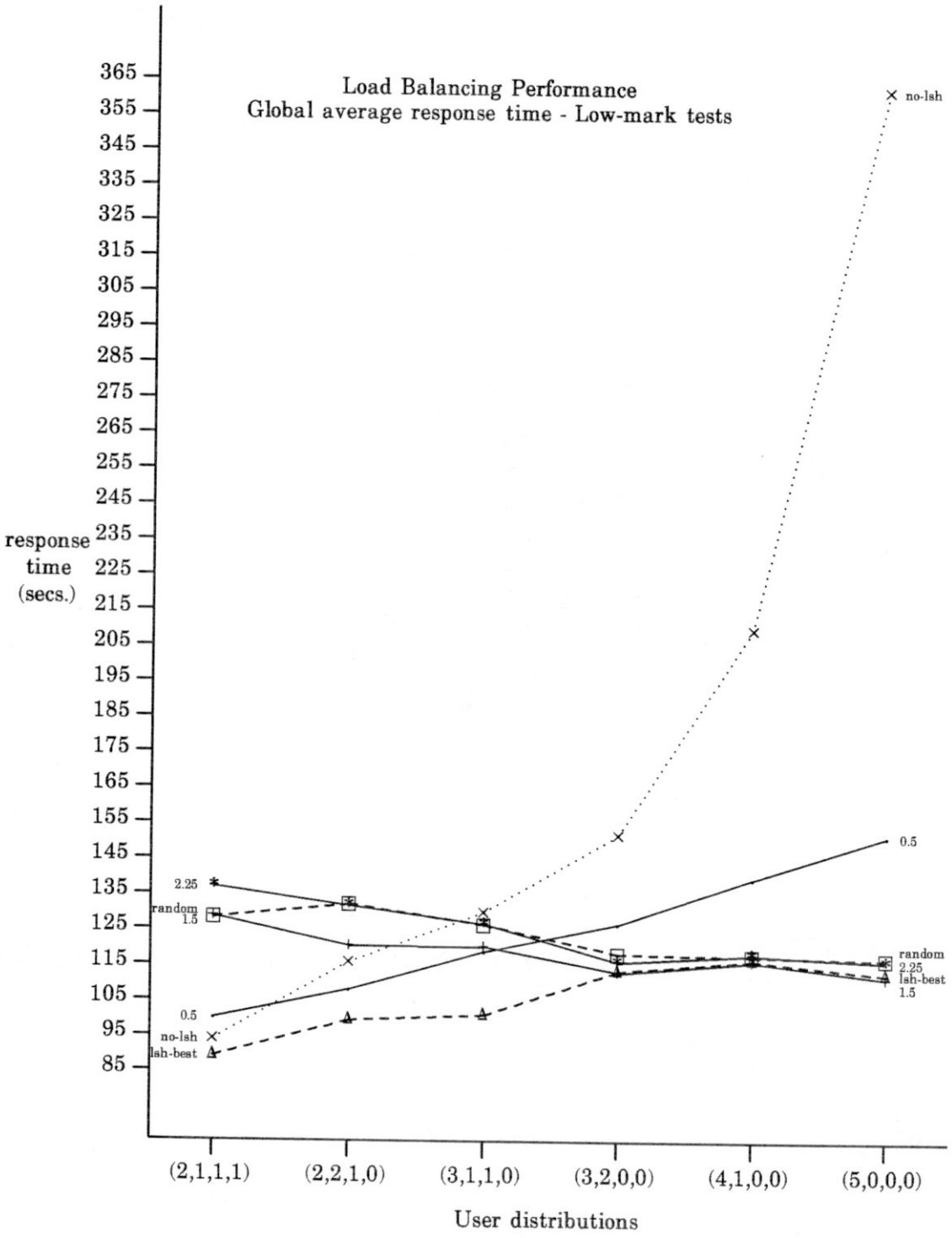

Figure 6: Response time of user jobs
using the Low-mark scheme

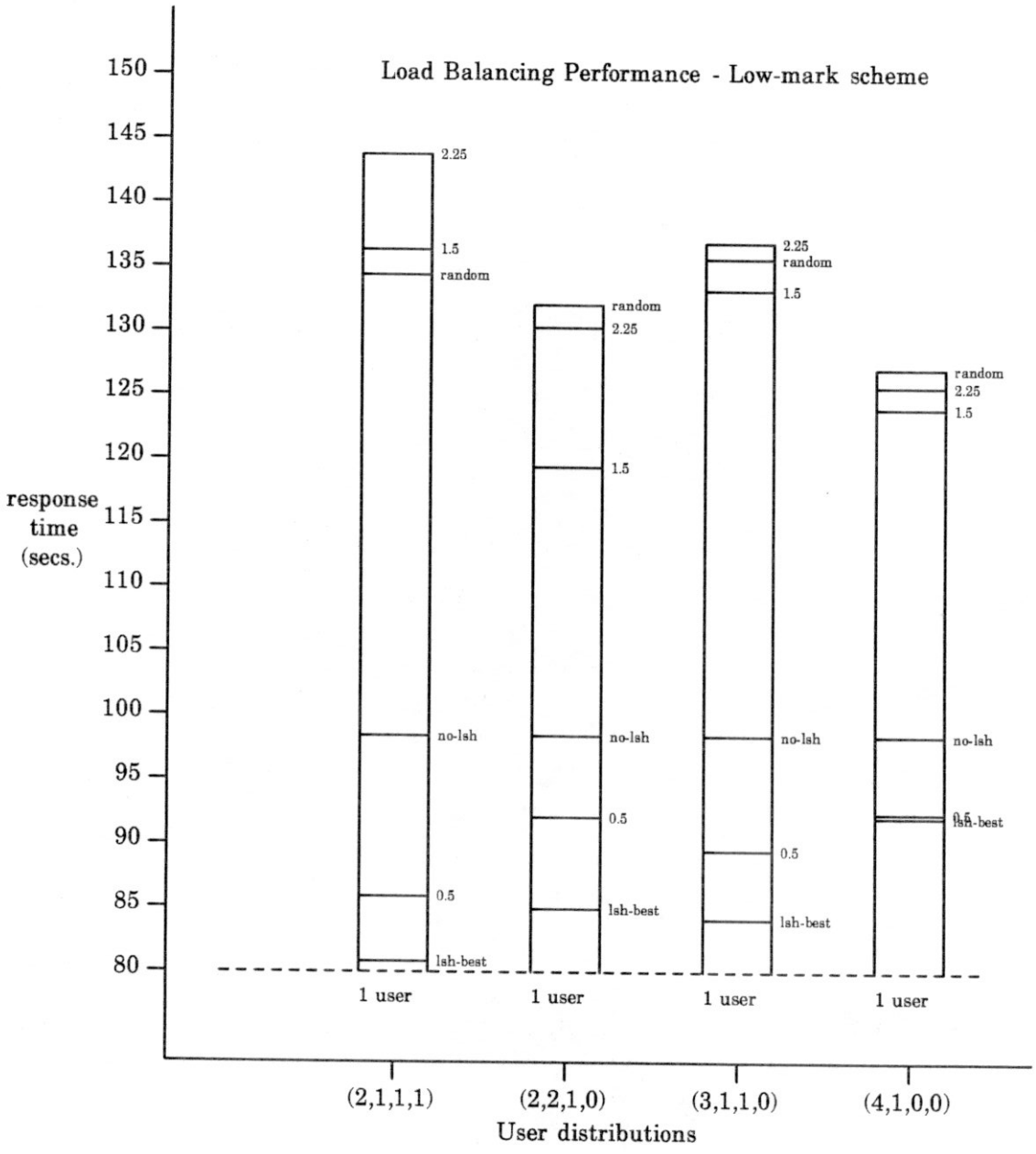

Figure 7: Local response time for
the least loaded nodes using Low-mark

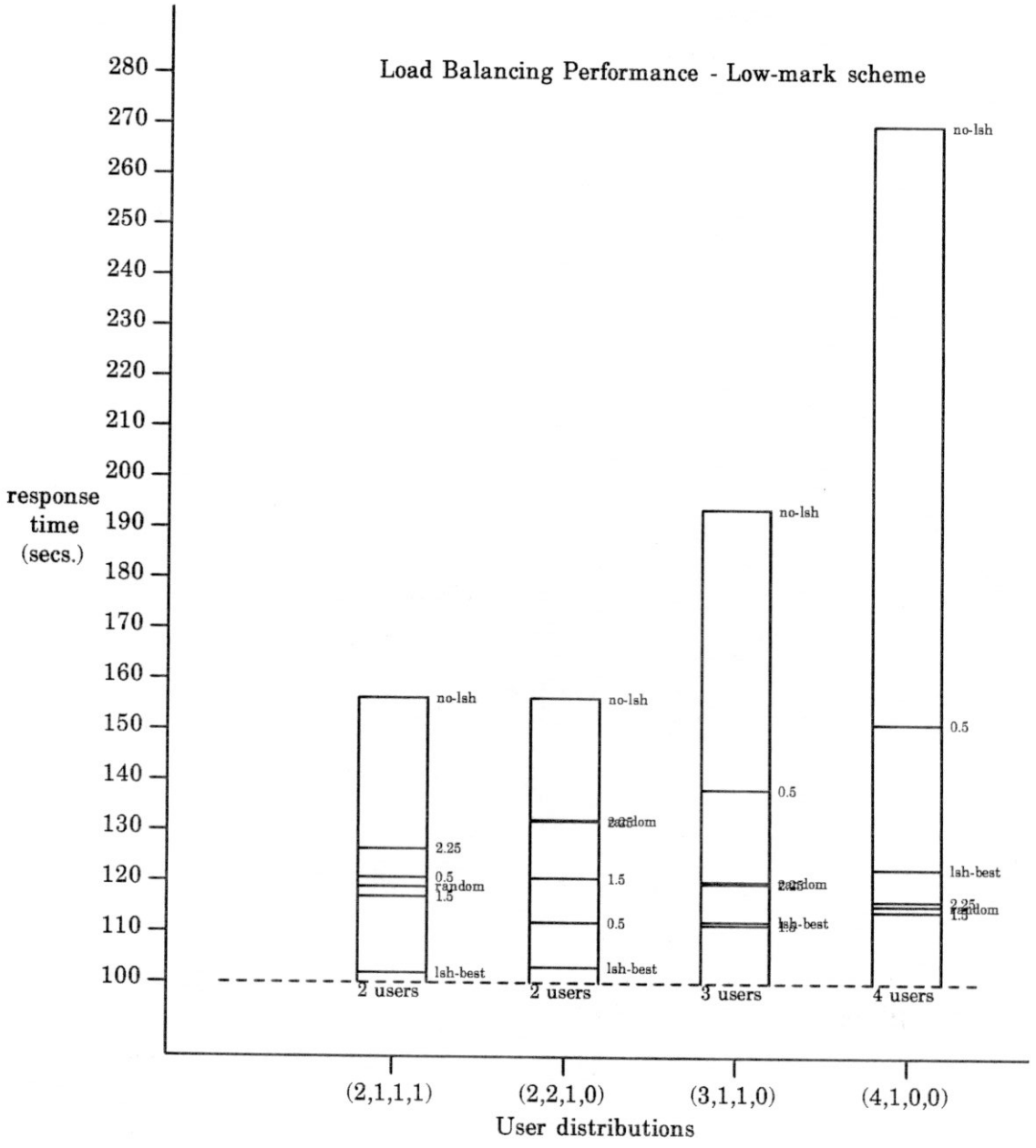

Figure 8: Local response time for
the most loaded nodes using Low-mark

4. A scheme to handle mixed computational environments

So far we have discussed each environment separately, but it is likely that distributed systems evolve to become a mixture of both environments: supercomputers dedicated to intensive computations, mainframes taking care of "housekeeping" operations, minicomputers acting as specialized servers and workstations dedicated to personal use. What is desired is a load balancing scheme that minimizes the global average response time for jobs being submitted to shared computational facilities, i.e., those belonging to the whole user community of the network, while making sure that resources owned by a particular user or group are not abused by other users of the system.

We believe that a scheme similar to lsh incorporating High-mark and Low-mark policies will be adequate for mixed environments. The idea behind the scheme will be to use the lsh mechanism as a hint to select the remote sites, and to use the High-mark and Low-mark policies to maintain the ownership of resources.

The above scheme could also be used when the overhead for broadcasting information becomes too high. As figure 9 shows there is an upper limit in the number of broadcast messages per interval of time that a machine can effectively process. This is generated by the overhead a node incurs in processing broadcast messages. To produce the figure we took four workstations and made them send broadcast messages at different time intervals. Two other workstations received these messages and processed them by updating their local information about the network. By using different intervals of broadcasting time, we produced different rates of messages sent per second. At each receiving node we measured the number of messages received per second and the amount of CPU time used in processing these received messages. Note that the same effect could be achieved if instead of decreasing the interval of broadcasting time we increased the number of machines transmitting. For example, 20 messages received per second can be achieved by having 20 nodes broadcasting at intervals of 1 second or by 400 computers transmitting every 20 seconds.

In networks with many nodes, it would be preferred to use a large interval of time to allow smaller processing overhead for the load balancing mechanism. This larger interval would have a direct effect on the accuracy of the information each node keeps. By using our scheme, the effect of this stale information would be ameliorated, thus reducing the amount of bad decisions that the load balancing strategy could make.

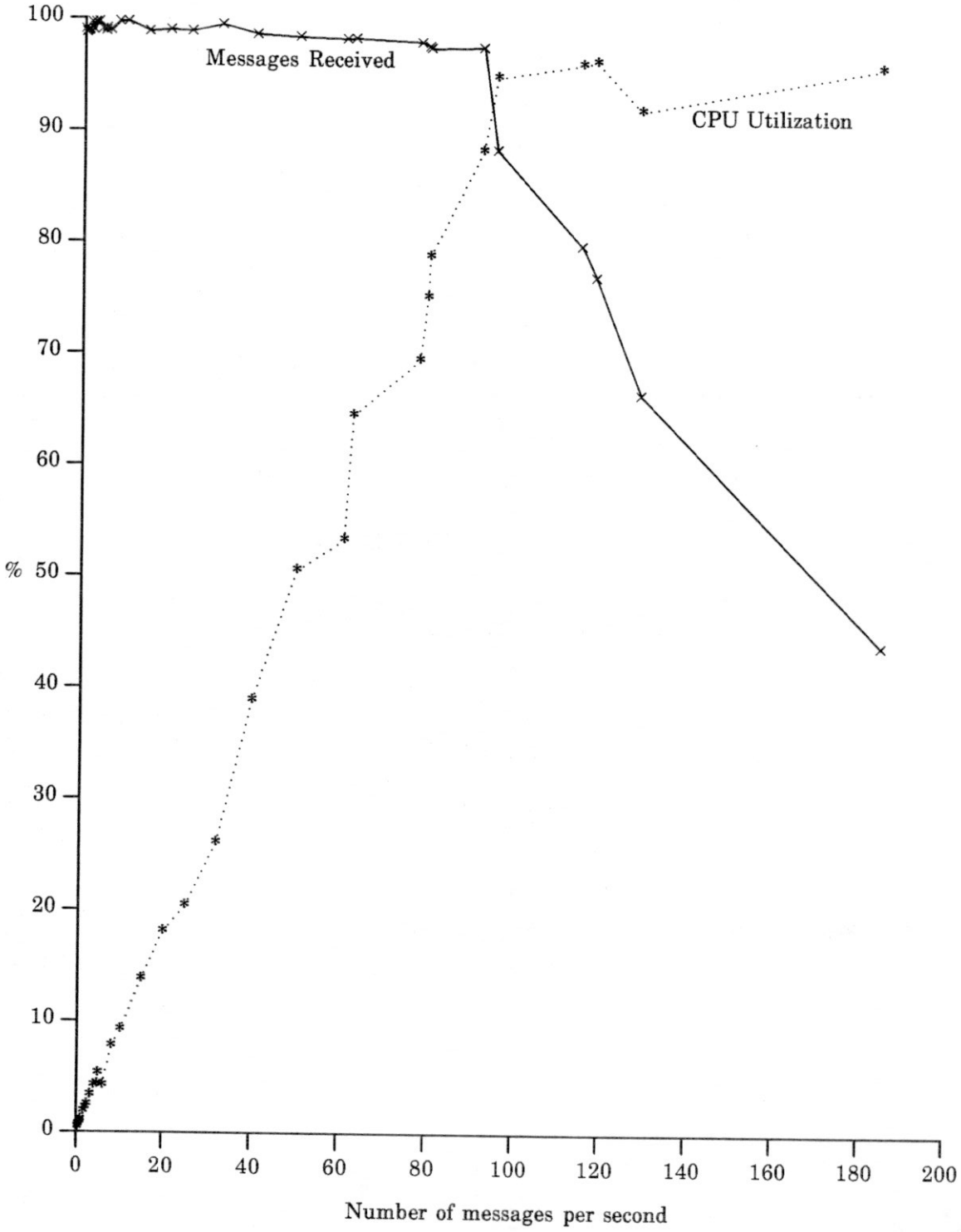

Figure 9: Processing overhead due to received broadcast message

5. Conclusions and future research

We discussed two known distributed environments where load balancing may improve performance: *pool of processors* and *independently owned processors*. We argued that because of their distinctive objectives different schemes are needed to handle load balancing in these environments. For the former, schemes should improve global performance indices since the network belongs to the whole user community. For the latter, schemes should make sure that the local response time of user's jobs at any node does not deteriorate more than a certain level.

We presented a comparison between an implemented version of a random scheme and a "least loaded" scheme showing the effectiveness of the "least loaded" scheme to the pool of processors model.

We discussed the principles that have to guide the load balancing schemes when dealing with independently owned processors. Mainly, we discussed the High-mark and Low-mark policies, and showed how these principles allow a processor to participate in load balancing schemes while maintaining the ownership of its resources. We suggested that a load balancing scheme that incorporates the High-mark and the Low-mark policies would be adequate in a network of workstations. A further point is that the capability to migrate jobs after they have started executing in a machine may be desirable for this environment. For example, once a machine gets overloaded by remote jobs, it could move some of these jobs to their originating nodes or to a new node. In this way control of the local resources could still be maintained.

Finally, we commented on the plausibility of a scheme to handle mixed computational environments based on an implementation of the "least loaded" policy combined with High-mark and Low-mark policies.

As for future research, we will concentrate in studying the independently owned processors environment. We have proceeded so far to test the High-mark and Low-mark schemes separately to avoid influence of one scheme over the results of the other. We will explore how these two schemes could be effectively used together to control the load sharing in a network of workstations. First, we will implement and test this combined scheme. Having a real user community, like our computer science department, using a load balancing scheme will give us an environment where the effect of load balancing can be studied. Second, we will work on the design of an algorithm for dynamically setting the High-mark and the Low-mark parameters, as suggested in section 3. Our goal is to formulate a scheme that using specifications submitted by the machine owner as well as the

current system load, will adapt the High-mark and the Low-mark parameters to achieve the best possible performance out of the load balancing scheme.

Recently, many multiprocessor workstations have been introduced to the market. In a network of such workstations, load balancing has to be done at two levels: within the local processors of a workstation and among the workstations. This presents a new interesting research problem.

There are many other interesting issues to explore in the area of dynamic load balancing. A couple of examples follow: What scheme, if any, is suitable for point-to-point networks? How could the design of load balancing schemes change when the capability to migrate partially executed jobs exists?

Acknowledgements

I would like to thank my advisor, Professor Rafael Alonso, for his invaluable support and advice. Also my gratitude to Peter Potrebic and Phil Goldman who designed and implemented the lsh software, to Ann Takata who produced the overhead measurements, and to Steve Kugelmass whose help improved the readability of this manuscript.

References

Alonso1986a.

Alonso, Rafael, Goldman, Phillip, and Potrebic, Peter, "A Load Balancing Implementation for a Local Area Network of Workstations," *Proceedings of the IEEE Workstation Technology and Systems Conference*, March 18-20, 1986a.

Alonso1986b.

Alonso, Rafael, "The Design of Load Balancing Strategies for Distributed Systems," *Proceedings of the U.S. Army Research Office Future Directions in Computer Architecture and Software Workshop*, May 5-7, 1986.

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.

Chow1979.

Chow, Y. C. and Kohler, W, H., "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," *IEEE Transactions on Computers*,

vol. C-28, no. 5, pp. 354-361, May 1979.

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.

Farber1972.

Farber, D. S. and Larson, K. C., "The System Architecture of Distributed Computer System: The Communication System," *Proc. of the Symposium on Computer Networks*, Brooklyn, Apr. 1972.

Hagmann1986.

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

Huang1986.

Huang, Hsu Chi-Yin and Liu, Jane W. S., "Dynamic Load Balancing Algorithms in Homogeneous Distributed Systems," University of Illinois at Urbana-Champaign Department of Computer Science Technical Report No. UIUCDCS-R-86-1261, 1986.

Krueger1984.

Krueger, Phillip and Finkel, Raphael, "An Adaptive Load Balancing Algorithm for a Multicomputer," Computer Sciences technical Report, University of Wisconsin - Madison, April 1984.

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.

Ni1985a.

Ni, Lionel M. and Hwang, Kai, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 5, pp. 491 - 496, May 1985.

Ni1985b.

Ni, Lionel M., Xu, Chong-Wei, and Gendreau, Thomas B., "A Distributed Drafting Algorithm for Load Balancing," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 10, pp. 1153 - 1161, October 1985.

Stankovic1984.

Stankovic, John A., "Simulations of Three Adaptive, Decentralized Controlled,

Job Scheduling Algorithms," *Computer Networks*, vol. 8, pp. 199-217, North-Holland, 1984.

SUN1986.

Sun Microsystems, Inc., "Networking on the Sun Workstation," *Distribution manual, part No: 800-1324-03 revision B*, 17 February 1986.

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.

Wang1985.

Wang, Yung-Terng and Morris, Robert J. T., "Load Sharing in Distributed Systems," *IEEE Transactions on Computers*, vol. C-34, no. 3, pp. 204 - 217, March 1985.

Williams1983.

Williams, Elizabeth, "Assigning Processes to Processors in Distributed Systems," *IEEE Parallel Proceedings*, pp. 404-406, 1983.

Zhou1986.

Zhou, Sognian, "A Trace-Driven Simulation Study of Dynamic Load Balancing," *Tech. Rept. No. UCB/CSD 87/305*, University of California, Berkeley, September 1986.

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.