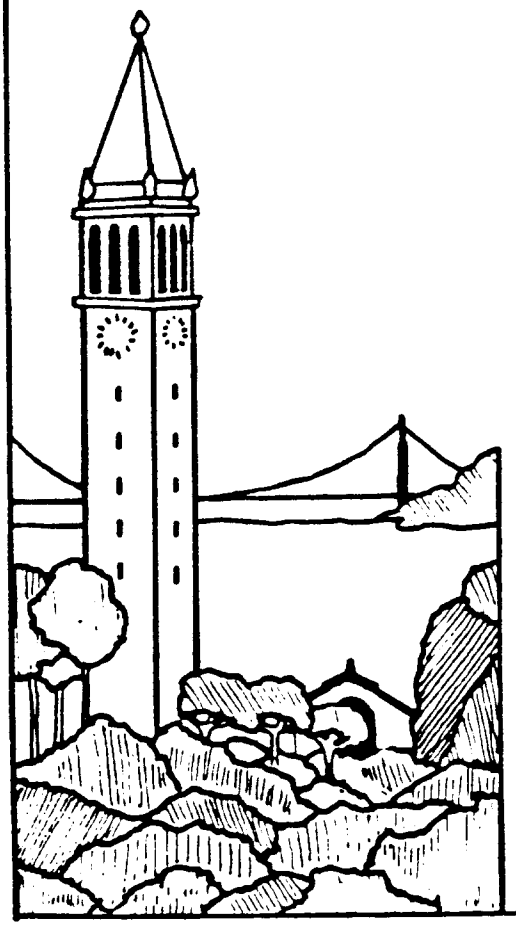
A Trace-Driven Simulation Study of Dynamic Load Balancing

Songnian Zhou



Report No. UCB/CSD 87/305

September 1986
PROGRES Report No. 86.4

Computer Science Division (EECS)
University of California
Berkeley, California 94720

A Trace-Driven Simulation Study of Dynamic Load Balancing

Songnian Zhou

Computer Systems Research Group
Computer Science Division, EECS
University of California, Berkeley †

ABSTRACT

A trace-driven simulation study of dynamic load balancing in homogeneous distributed systems supporting broadcasting is presented. We use information about job CPU and I/O demands collected from a production system as input to a simulation model that includes a representative CPU scheduling policy and considers the message exchange and job transfer costs explicitly. Seven load balancing algorithms are simulated and their performances compared. We find that load balancing is capable of significantly reducing the mean and standard deviation of job response times, especially under heavy system load, and for jobs with high resource demands. The performances of all hosts, even those originally with light loads, are generally improved by load balancing. The reduction of the mean response time increases with the number of hosts, but levels off at around 30 hosts. Algorithms based on periodic or non-periodic load information exchange provide similar performance, and, among the periodic policies, the algorithms that use a distinguished agent to collect and distribute load information cut down the overhead and scale better. They are also the most appropriate algorithms for adaptive load balancing, which has the potential of offering near-optimal performance under a wide spectrum of system configurations and load conditions. System instability in the form of host overloading is possible when the load information is not up-to-date and the system is under heavy load; however, this undesirable phenomenon can be alleviated by simple measures. Load balancing is still very effective even when up to half of the eligible jobs have to be executed locally. The trace-driven simulation approach to the study of load balancing is found to be efficient and effective, and is recommended for use before implementation efforts.

† This work was partially sponsored by the Defense Advanced Research Projects Agency (DoD), Arpa Order No. 4871, monitored by Space and Naval Warfare Systems Command under Contract No. N00039-84-C-0089, and by the National Science Foundation under grant DMC-8503575. The views and conclusions contained in this document are those of the author and should not be interpreted as representing official policies, either expressed or implied, of the Defense Research Projects Agency or of the US Government.

1. Introduction

Distributed computer systems are becoming increasingly available because of the drop in hardware costs and advances in computer networking technologies. An important advantage of distributed systems is the potential of resource sharing to provide the users with a rich collection of resources that are usually unavailable or highly contended for in stand-alone systems. Examples of sharable resources are files, computing power, and printers. It is frequently observed that, in a computing environment with a number of hosts connected by networks, there is a high probability that some of the hosts are heavily loaded, while others are almost idle. Even if the hosts are evenly loaded over long periods, such as half an hour or more, the instantaneous loads are likely to be fluctuating constantly. † This suggests that performance gains may be achieved by transferring jobs from the currently heavily loaded hosts to the lightly loaded ones. This form of computing power sharing, with the purpose of improving the performance of a distributed system by redistributing the workload among the available hosts, is commonly called *load balancing*, or *load sharing*. ‡

The problem of load balancing has been studied using a number of different approaches over the years. The early works mainly concentrated on static load balancing [3, 17, 18, 21]. In those studies, job transfer decisions are made deterministically or probabilistically without taking into consideration the current state of the system. The problem of program module assignment has also been studied in a number of forms, with the basic assumption that the program concerned can be partitioned into a number of modules with known resource consumptions and inter-module communication costs. Load balancing is formulated as a mathematical programming or network flow problem, and solved by optimizing some performance index such as the average response time or the resource utilizations.

Static load balancing is simple and effective when the workload can be sufficiently well characterized beforehand, but it fails to adjust to the fluctuations in system load. In contrast, dynamic load balancing* attempts to balance the system load dynamically as jobs arrive. Because of its generality and ability to respond to temporary system unbalances, dynamic load balancing has received increasing attention from the research community [7, 8, 9, 11, 15, 16, 20]. Livny and Melman [16] showed, using simple queuing network models and simulation, that dynamic load balancing can greatly improve average job response time. They also proposed a number of implementable algorithms for load balancing. Eager et al. [9] carried the work further by systematically studying a number of dynamic load balancing algorithms with different levels of complexity. Their results confirmed the great potential of load balancing. They also claimed that relatively simple algorithms can provide substantial performance improvements, while more complicated algorithms are not likely to offer much further improvement. Wang and Morris [20]

---

† Such observations, of course, are dependent on the system and the applications being run. For instance, in a main-frame batch data processing environment, the loads might be even over long periods of time. In contrast, however, in a workstation-rich environment, which is becoming more and more popular, the probability of a majority of the stations being idle or almost idle is very high [19].

‡ The term load balancing has sometimes been used to imply the objective of equalizing the loads of the hosts, whereas load sharing simply means a redistribution of the workload. We will use the term load balancing in the rest of this paper, but without the stronger connotation.

* Some authors used the terms *adaptive load balancing* and dynamic load balancing interchangeably. We decided, however, to reserve the former for a particular form of load balancing to be described later in this paper.

conducted a comprehensive study and pointed out that the choice of a load balancing algorithm is a crucial design decision. They also proposed a performance metric called the Q-factor, and used it to evaluate the quality of the algorithms. Leland and Ott [15] performed an extensive study of process behavior in the VAX/UNIX environment and evaluated the usefulness of initial process assignment and process migration as forms of load balancing. A number of other researchers have also considered process migration in their load balancing algorithms [1, 4]. To limit the scope of our study to a manageable level, however, we will not consider process migration in this paper. Process migration is also much more difficult to implement, and involves higher costs in most systems.

Although different authors make very different assumptions about system structures and overhead costs, the main tools of study in dynamic load balancing have been queuing network models and simulation with probabilistic assumptions about job arrivals and resource demands. Unfortunately, a reasonably accurate analytic model for a real-world system with a load balancing scheme of modest complexity can be very difficult to construct. Solving the models is even harder. Consequently, many researchers are forced to make simplifying assumptions that are often unrealistic, rendering the results of the studies subject to suspicion. For example, in order to make the model tractable, the job interarrival time and the job execution time are often assumed to be exponentially distributed. The utilizations of the hosts are sometimes assumed to be the same, and the effects of the system scale on load balancing performance are often ignored. For similar reasons, the costs of exchanging load information and other types of costs associated with load balancing are often ignored or grossly simplified. Simulation models driven by probability distributions are capable of handling greater system complexity and thus solving a larger class of problems, but it is still unclear how much error in the results is introduced by the distributional assumptions made by the investigators.

To substantiate these criticisms, we traced a production VAX/UNIX* system for a number of extended periods during working hours, and recorded the arrival times of the processes†, as well as their CPU and disk I/O demands. The distributions of these measurements are shown in Figures 1, 2 and 3, respectively. It can be seen that none of them follow an exponential pattern. The inter-arrival time distribution is not very far from exponential, whereas the CPU and I/O demand distributions are both highly skewed ‡. Similar observations have been made by other researchers [5, 6, 15].

In this paper, we study the problem of dynamic load balancing using an approach different from those mentioned above. Job traces collected from a production system are used to drive a simulation program that implements a number of load balancing algorithms. In this way, we eliminate the errors caused by assumptions about the workload. The costs of message exchanges and job transfers are considered so that performance comparisons between the algorithms can be made on an equal basis. Two broad categories of algorithms are commonly recognized. In *source*

---

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

† In a UNIX system, a *job* corresponds to a command line input by a user, and a number of *processes* may be created to carry out the job. We will not insist on this distinction in this paper, however.

‡ For a job's I/O demand, both synchronous and asynchronous disk I/O's are considered, while disk cache hits are properly excluded.

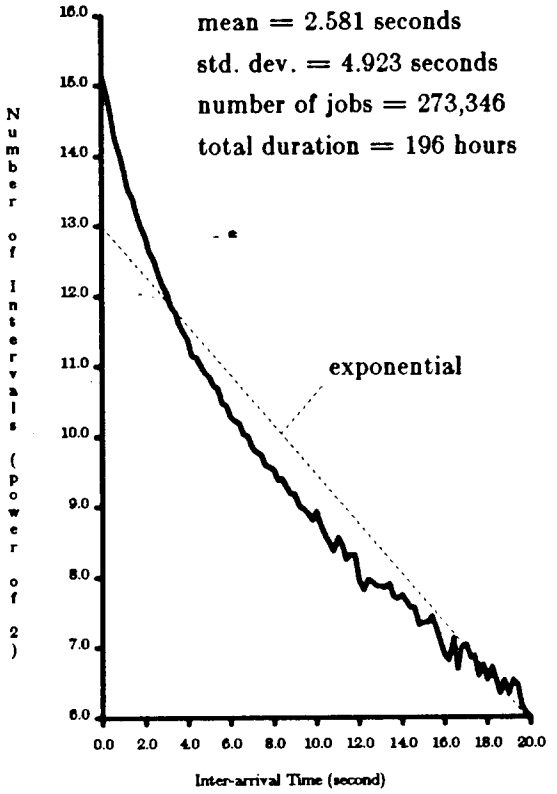


Figure 1. Distribution of job inter-arrival times.

initiative algorithms, the hosts where jobs arrive take the initiative to transfer the jobs, whereas in *server initiative* algorithms, hosts able and willing to receive transferred jobs go out to find such jobs. A host may well be a source and a server at the same time. We concentrate on source initiative algorithms in this paper.

A load balancing algorithm consists of a number of components.

(1) The *information policy* specifies the amount of load and job information made available to job placement decision maker(s), and the way by which the information is distributed. We may require that the loads of all the hosts in the system be available to the decision maker(s). Alternatively, no or only partial information may be available. Periodic updates may be used to distribute load information, or the information may be provided upon request (demand-polling). A distinguished agent may be involved in the load information distribution, or no such agent may exist.

(2) The *transfer policy* determines the eligibility of a job for load balancing based on the job and the loads of the hosts. It may not be desirable, for example, to transfer small jobs, and some jobs may require specific resources available only on certain hosts, thus being unsuitable for consideration.

(3) The *placement policy* decides, for eligible jobs, the hosts to which the jobs should be transferred. An attempt may be made to select the least loaded host in the system, or only an acceptable host is sought so that less load information is needed. If no suitable host can

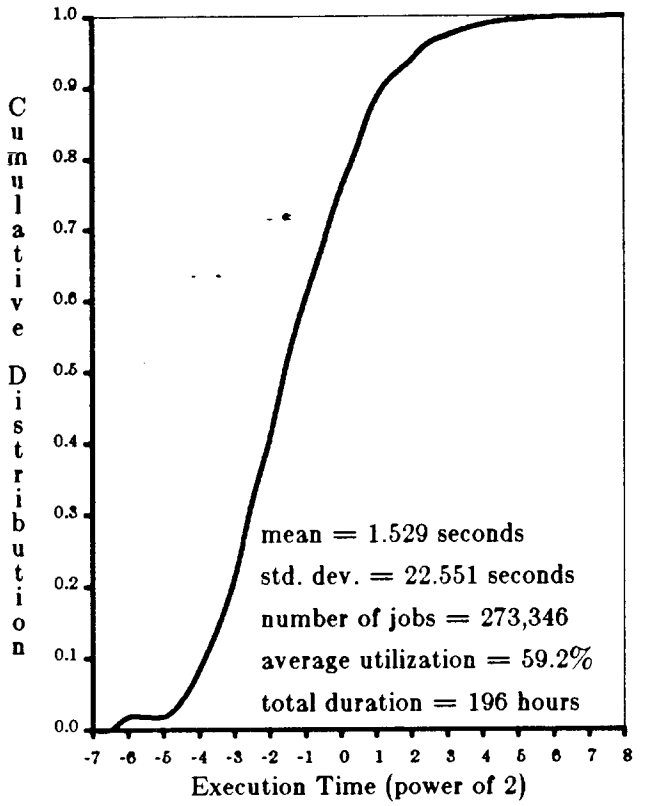


Figure 2. Cumulative distribution of job CPU times (in seconds).

be found, the jobs will have to be processed locally.

The above three component policies of a load balancing algorithm are not isolated from each other, but interact in various ways. For example, the load information available limits the possible transfer policies. Because of the large number of options for each component policy, it is impossible to study all possible policy combinations in this paper. Instead, we shall concentrate on the information policies and some of the related placement policies, while keeping the other aspects of the scheme fixed. Specifically, we are interested in comparing the performances of the algorithms using periodic updates and of those acquiring information on demand. For the periodic policies, we want to evaluate the performance impact of a global agent that collects and distributes load information of all the hosts in the system. We also want to study the problem of instability caused by a number of hosts sending jobs all at the same time to a lightly loaded host, thus making it overloaded. A number of representative load balancing algorithms are defined and studied in detail. However, our objective is not to select the best algorithm, but rather, to study the characteristics of various types of algorithms and the tradeoffs between conflicting requirements.

The important results from this study include the following:

- A load balancing scheme using any reasonable algorithm can improve the job response times by 30-60%, and make them much more predictable.

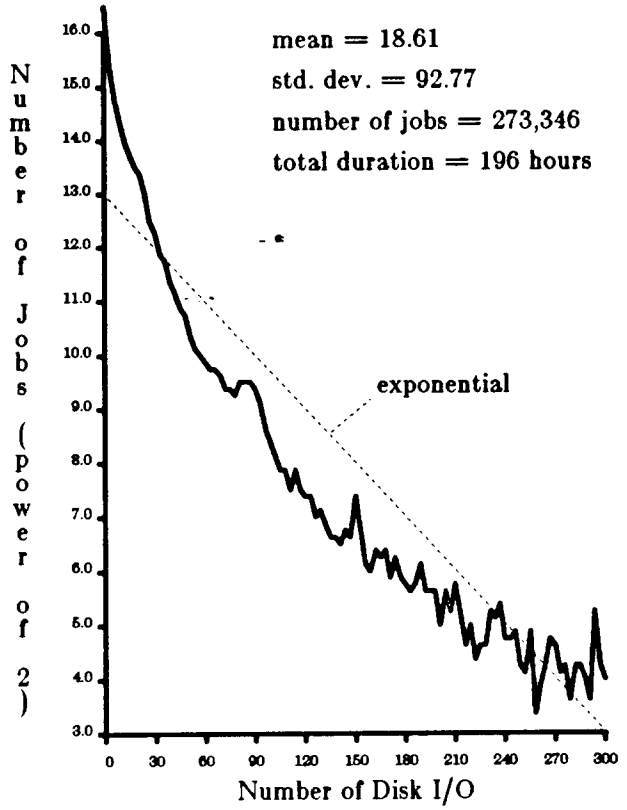


Figure 3. Distribution of the number of disk I/O's per job.

- The mean response times of jobs on *every* host, even on those originally with light loads, are reduced by load balancing.
- Periodic and non-periodic information policies provide comparable performance.
- For the periodic information policies, the global algorithms impose less overhead on the system than the distributed ones (typically half or less for systems with 20 or more hosts), and, hence, can support larger systems.
- Greater performance improvement can be gained by increasing the system size, but the improvement levels off beyond a few tens of hosts, at which point it becomes more advantageous to implement load balancing in clusters.
- Instability may occur when load information is stale and the system load is high, but it can be alleviated by simple measures.
- Load balancing can still be highly effective when up to half of the jobs that are otherwise eligible for load balancing must be executed on their local hosts.

Our study also provides insights into the choice of a load balancing algorithm under different system environments and load conditions.

In the next section, we describe the system we simulate and the structure of the model. We also discuss the load and performance indices we use. Section 3 describes the algorithms that we studied in the simulation. The simulation results are presented in Section 4, along with a discussion and comparison of the algorithms. Some concluding remarks are made in Section 5.

2. Experiment Design

2.1. The Job Trace

A distinguishing feature of our study is the use of job traces instead of probability distributions to describe the arrival times and resource demands of the jobs. We traced a production VAX-11/780 system running Berkeley UNIX to collect job traces consisting of tuples of the format

$$<\textit{job arrival time, CPU time demand, number of disk I/O's}>.$$

Previous measurement studies conducted by the author [22] show that the CPU is the most contended resource in the type of time-sharing systems from which the job traces are derived. There is usually plenty of main memory, hence little paging and almost no forced process swapping occur. The networking subsystem is not heavily loaded either. Therefore, we will consider only CPU and disk I/O in our model, while retaining confidence in the results of the simulation.

Heterogeneity, either architectural or configurational, complicates the load balancing problem greatly, and is a deviation from the primary concerns of this research. Therefore, we will concentrate on homogeneous systems. In fact, to insure homogeneity and to ease the trace collection efforts, sessions of job traces were collected on the *same* host at different times to represent a number of hosts connected by a network. † The selection of simulation session length is important because the boundary effects caused by jobs started before the session begins and by those finishing after the session ends may significantly affect the accuracy of the results. On the other hand, longer sessions involve greater efforts in trace collection and simulation. We chose the length of each session to be four hours. Typically, about 6000 processes are created on each host during this period. Even so, some of the processes executing during a session are not included. Such processes are mostly system services that are started at system boot time and run until the system goes down, and a few very long batch jobs. Though small in number, they can represent a significant portion of CPU time consumption. As a result, the simulated CPU utilizations during the sessions are lower than in reality, typically by 5-15 percent.

2.2. Model Structure

The simulation model structure is shown in Figure 4. We adopt a foreground-background round-robin scheduling policy for the CPU. The time quantum is 100 milliseconds, the same as that used in the Berkeley UNIX system from which the trace was derived. After a job has accumulated 500 milliseconds of CPU time, it is put into the background queue, which will be checked only if no job is available in the foreground queue. Since about 60-65% of the jobs have execution times below this threshold, they will not be sent to the background queue, thus receiving priority service. While the CPU scheduling policies in computer systems are usually more complicated, we feel that the above policy captures their essential features, and may be considered representative. Since the level of contention at the disks is usually low under normal operating conditions in the type of system we measured [22], we model them as infinite servers causing only processing delays,

† It is recognized that, by so doing, the possible temporal correlations between the loads of the various hosts are lost.

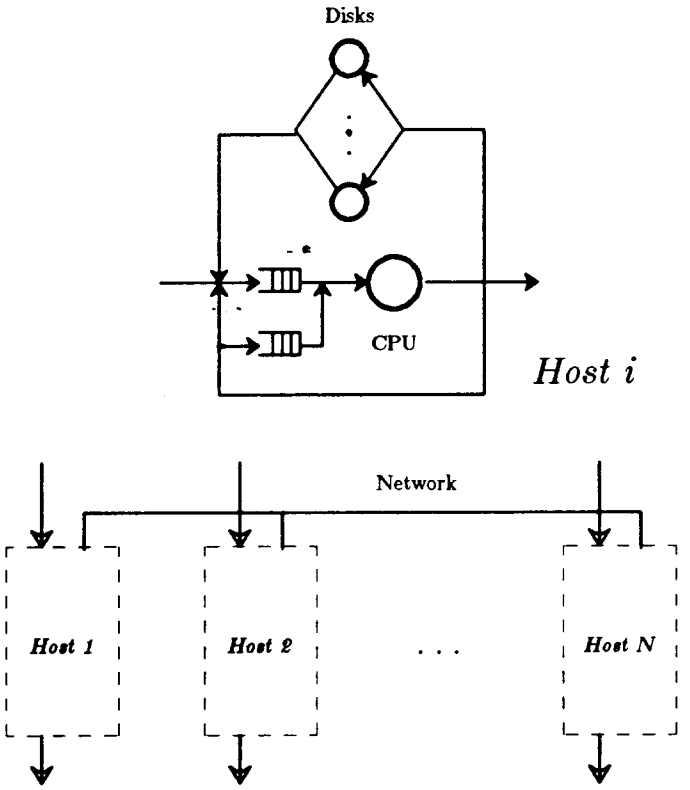


Figure 4. Structure of system used in simulation.

but no queuing delays. I/O operations are assumed to be evenly spread throughout the execution of the job ‡, and each disk I/O is assumed to take 30 milliseconds. A communication network permits message passing and job transfers between the hosts. Since we are most interested in load balancing in local distributed systems, we assume that the underlying network supports broadcast (e.g., Ethernet). We also assume the existence of a distributed file system so that the the costs of accessing the program and data files are roughly the same for all of the hosts. As a result, the files do not have to be moved with the jobs to be load balanced. This assumption will be increasingly appropriate for future systems designed for distributed computing. Since our trace data is derived from a time-sharing system without the support of a distributed file system, we are unable to simulate the contention at the file servers, and we also do not have measurement data on remote file accesses. The cost of 30 milliseconds for an I/O operation is therefore a rough approximation.

‡ Recording the times of the I/O operations during job execution would greatly complicate our trace collection effort and the model construction and simulation, without providing significant benefit, in terms of model accuracy, since the disks are not the points of contention.

2.3. Cost Assumptions

There are basically two types of overhead costs involved in load balancing. First, current load indices of the hosts have to be computed and messages exchanged to make them known to the decision makers. Secondly, placement decisions need to be made and jobs transferred between the hosts. CPU time and network bandwidth are consumed for these purposes. The latter of overhead also directly introduces extra delays in the jobs involved. (So is the former if load information is acquired while the job to be balanced is waiting, as is the case with a number of algorithms to be studied.) It has been experimentally observed that, in most current installations, local area networks, such as the Ethernet, usually have plenty of bandwidth, and the delays in the network are small compared to the CPU cost of executing the communication protocols [14]. Consequently, we only consider CPU time overhead in this study. We assume that message exchange and job transfer have preemptive priority over job execution. Based on measurements from our experimental implementations of load balancing on the VAX/UNIX and SUN/Unix machines, we assume that computing the current load and sending it out takes 20 milliseconds of CPU time, while receiving load information and processing it takes 10 milliseconds. A job transfer is assumed to take 100 milliseconds of CPU time for both the sending and the receiving host, and causes 200 milliseconds delay to the job being transferred. This assumption seems to be less critical than that for message cost because the algorithms we study mainly differ in their information policies; a change in job transfer cost is likely to change their performances by similar amounts.

It should be pointed out that the above cost assumptions are very approximate; the actual costs in terms of the CPU times spent and the job delays introduced are highly sensitive to the load conditions of the hosts involved and the network load. They are also dependent on the implementation of the underlying system, as well as on the size of the message and on that of the job.

2.4. Load and Performance Metrics

In order to compare the performances of various load balancing algorithms, we need a number of metrics. First, it is important to characterize the load on the whole system, as the performance of load balancing schemes varies with the system load. We choose the average CPU utilization of all the hosts over the entire session as the *load level indicator* since it represents the level of contention for the most critical resources in the system. We are also interested in a *load index* that we can use to predict the response time of a job if that job is executed on a particular host. Ferrari [10] pointed out, using mean value analysis, that a linear combination of the resource queue lengths in a computer system can be an excellent predictor of job response time, with the coefficients being the estimated resource consumptions of the job. In a previous measurement study [22], we found that the CPU queue length has a high correlation with the job response time in a CPU-bound host, and hence suggests itself as a good load index.

To measure and compare the effectiveness of load balancing algorithms, we need to define a *performance index*. We choose the mean job response time because decreasing the job response time is our primary objective of load balancing. However, this does not measure the variability of the job response times. We will use the standard deviation of the response times of all the jobs to complement the mean response time.

3. Load Balancing Algorithms

We studied seven algorithms that use different types of information policies and related placement policies. For ease of comparison, we base the transfer policy of all the algorithms on the local host load and job execution time thresholds. When the CPU queue length of a host is at or below a threshold, all jobs arriving there are processed locally. Otherwise, all the jobs arriving at that host and with execution times above a certain threshold are *eligible* for load balancing. Although job execution times are difficult to predict, it is possible to classify the jobs into two rough categories: "big" jobs which are worth considering for load balancing, and "small" jobs not to be considered. Moreover, estimation errors can be easily tolerated, as long as they are not too frequent. Our studies of jobs submitted over 30 days show that such a classification can be made with a very high success rate simply by looking at the job names. For example, a text processing job will almost certainly take over 1 second of CPU time, whereas a directory checking operation is clearly not worth considering for load balancing. One result of this research is that the performance of the load balancing algorithms is quite robust with regard to the job execution time threshold (See Section 4.4).

The following algorithms were studied:

GLOBAL

Every T seconds, one of the hosts, designated as the *load information center* (LIC), receives load updates from all the other hosts and assembles them into a load vector, which is then broadcast to all the other hosts. If the load of a host is the same as that sent out the last time, however, no update needs to be sent to the LIC. This applies to the next algorithm, DISTED, as well.

The placement policy of the GLOBAL algorithm, as well as that of the next algorithm, is as follows. The local version of the load vector is searched for a host with the shortest CPU queue length, and, if the difference in CPU queue length between the local host and the potential destination is at or above a given limit l (usually 1 or 2), the job is sent there. If there are several hosts with the same shortest queue length, which is often the case, the first one is selected. This rule, together with a randomized starting point for the search, can potentially alleviate the instability problem as we will discuss later.

DISTED

Instead of reporting the local load to a centralized LIC as in GLOBAL, each host broadcasts its load periodically for the other hosts to update their locally maintained load vector.

CENTRAL

In the above two algorithms, placement decisions are made by each host using the local version of the load vector. In the CENTRAL algorithm, there exists a central scheduler for all the hosts. When a host decides that a job is eligible for load balancing, it sends a request to the central scheduler, together with the current value of its load. The central scheduler selects a host with the shortest queue length and informs the originating host to send the job there. The load vector on which the scheduler bases its decisions is updated using only the load information sent by the hosts with the job requests.

CENTEX

The same as CENTRAL except that, periodically, each host sends its local load to the LIC (CENTral with EXchange). This algorithm can be regarded as a hybrid of GLOBAL and CENTRAL.

For the above four algorithms, the load vector used in the placement decision is updated by increasing the load of the destination host by an adjustable constant (currently 1). All the algorithms assume that the loads of all the hosts are known to the placement decision makers, with some delay. The algorithms below use less system state information, and thus have smaller overhead costs.

RANDOM

This algorithm uses minimum load information. When a job is found to be eligible for load balancing, it is sent to a randomly selected host. The receiving host treats the transferred job exactly as if it had arrived locally. To avoid the undesirable situation in which a job bounces around indefinitely, we set a limit L_t such that the L_t'th host receiving the job has to process it no matter what its load is.

THRHLD

A number of hosts up to a limit L_p are polled when an eligible job arrives, and the job is transferred to the first host whose load is below a fixed threshold. If no such host is found, the job is processed locally. When the message exchange cost is much lower than the job transfer cost, this algorithm wins over RANDOM by avoiding costly job transfers.

LOWEST

This is similar to THRHLD except that, instead of using a threshold for the placement, a fixed number of hosts are polled and the most lightly loaded host is selected. Thus, when message overhead is higher, a potentially better host may be selected than by THRHLD.

The last three algorithms above are identical to the ones studied by Eager et al. [9] However, we use a trace-driven simulation method to evaluate them, and we compare them to those algorithms that use a load vector. The algorithms above make placement decisions on the basis of various amounts of system state information. Since we consider the overhead costs of load balancing explicitly, a direct assessment of the appropriate amount of load information for load balancing can be made.

For comparison, we also implemented three boundary cases of load balancing:

NoLB

No load balancing is attempted; all arriving jobs are processed locally.

NoCOST

This is the unrealizable ideal case in which the current CPU queue lengths of all the hosts are known to the transfer decision makers at no cost (in terms of CPU time *and* job delay), and the transfers of jobs are also assumed to be costless.

PartCOST

This is the partly-ideal case in which perfect load information is assumed to be known at no cost, but job transfer costs are considered.

The performance of all the algorithms can be expected to be between those of NoLB and NoCOST.

There are a large number of potentially useful load balancing algorithms besides the ones listed above. As we stated earlier, our primary concern in this paper is not the particular algorithms to use, but rather the effects of different approaches to load information gathering and placement decision making.

4. Simulation Results

Simulation runs with various system sizes and load levels were executed. To make the performance comparisons between the algorithms meaningful, a number of simulation runs were conducted for each algorithm with different adjustable parameter values (e.g., job threshold, load exchange period), and the best response time was selected. In this way, the comparisons are between the best achievable performances of different algorithms, and it is hoped that they reveal the qualities of the algorithms. The results of the simulation experiments are presented in the following sections.

4.1. Comparison of the Algorithms

Figure 5 shows the average response times of a system of 28 hosts under the load balancing algorithms described above. Since job traces are used to drive the model, we cannot control the utilization of the system. However, it is essential to observe the performance of the algorithms under various load conditions. We achieve this by multiplying the job interarrival times by a constant factor. By varying the multiplication factor, we are able to generate a number of points for each algorithm. Although the job stream is altered, the job characteristics (i.e., execution time, number of I/O) remain the same. We feel that such a modification to the job stream is unlikely to introduce significant errors in the results. †

The first observation in Figure 5 is that all the algorithms provide substantial performance improvements over a wide range of system loads, compared to the NoLB case. In fact, response times reasonably close to those of the NoCOST case are achievable. The higher the system load, the greater the improvements. While we observed a greater-than-average improvement in the mean response time of big jobs (e.g., with execution times greater than 1 second) the mean response time of the small jobs does not suffer as a result. Figure 5 also demonstrates clearly the relative performances of the algorithms. The performance of CENTRAL is the worst of the seven. This is mainly because the global scheduler relies only on the load information provided with job scheduling requests. It is observed that the frequency of placement decision making for each host is one per 5-20 seconds, when the job threshold is 0.5-1.0 second. At such long intervals, the loads of the hosts are likely to have changed substantially. Consequently, a high percentage of the global scheduler's decisions are wrong.

In sharp contrast, the CENTEX algorithm, which is the same as CENTRAL except that load information is periodically reported to the LIC, provides the best performance among the

† Two other choices are to multiply the job execution times by a factor, and to use different job streams. However, they both alter the job characteristics and seem to introduce more changes to the workload than the method we used, thus making the comparison of performances under different workload levels less meaningful.

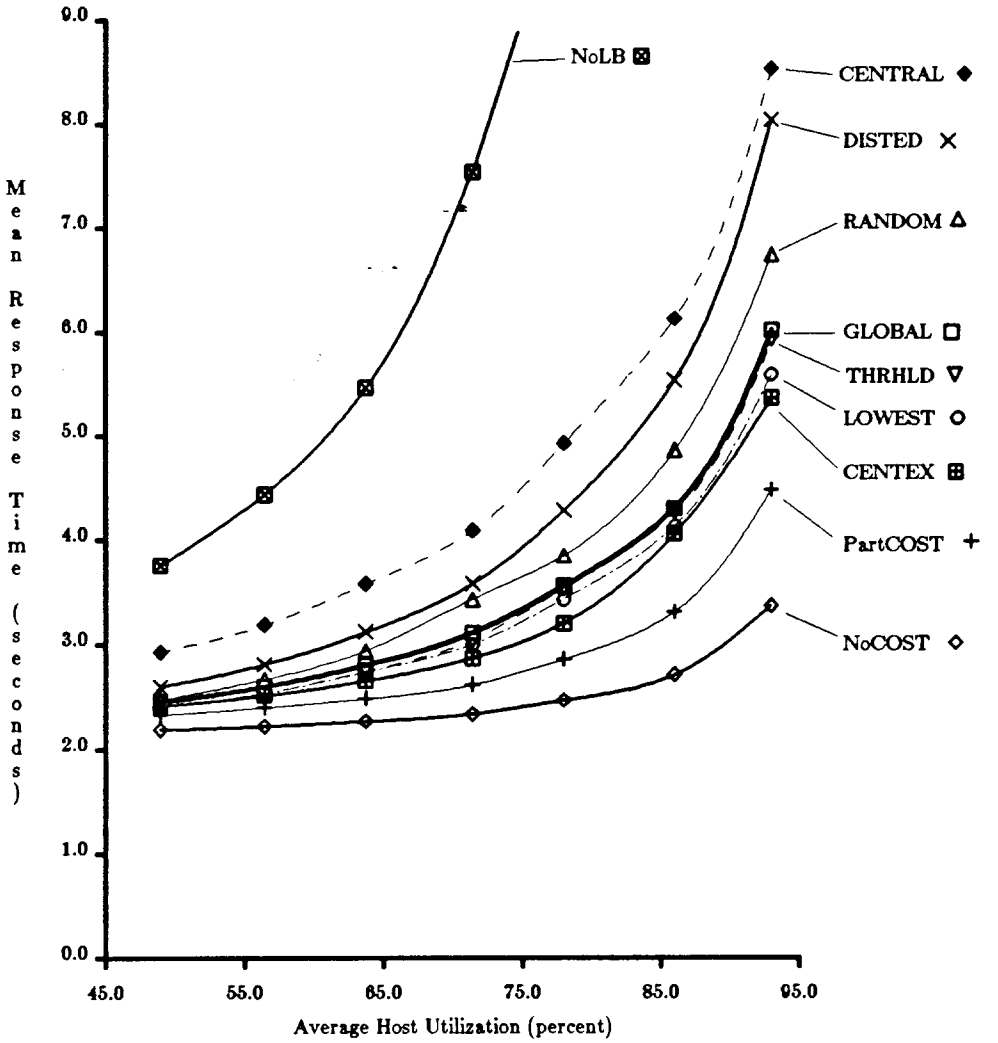


Figure 5. Average response times under different load levels (28 Hosts).

seven. It has been widely assumed that, in distributed systems, centralized solutions are undesirable because they tend to create performance bottlenecks and single points of failure. Such a view, however, may be too simplistic if unqualified. The best solution is environment and problem dependent. For load balancing, if the interprocessor communication is relatively efficient (such as the case in this paper), and the system scale is limited (up to 50-100 hosts), the centralized approach to load information distribution and job placement may be simple and efficient, as demonstrated by Figures 5 and 6. The costs of job placements is reduced for all the hosts except the LIC, as they now only need to send local load information and placement requests to the LIC, rather than maintaining system-wide load information and performing placements themselves. For the LIC, we observed that up to 35% of its CPU time may be spent for load balancing functions supporting a system of 49 hosts. Although this is a high overhead for this host, it is a small price to pay for the whole system. In return, excellent placement decisions based on up-to-date information are achieved. This explains why the performance of CENTEX is slightly better than those of THRHLD and LOWEST, which only attempt to select a host from a small subset of the

hosts. For many distributed applications, availability is crucial, hence a centralized solution is not appropriate. This is not the case with load balancing, however. If the LIC goes down, some other host can quickly detect the condition and take over its role. The loss of load information is not a serious problem because load information becomes obsolete in a short while anyway. The brief interval during which load balancing is unavailable should be easily tolerable because load balancing is not an essential system service such as the naming service; its absence should in no way interfere with system operations. In fact, an implementation using essentially the CENTEX algorithm has been reported to provide effective load balancing [12]. In that environment, inter-host communication is extremely fast, and the global scheduler is claimed to be able to process 1000 requests per second.

The comparison between the GLOBAL and DISTED algorithms is highly instructive. Since they are the same, except for their information policies, the significant performance difference reveals the advantages of using a global agent as a relay point for load information exchange. Assume that there are N hosts in the system, and let the update period be T seconds, and the cost of sending and receiving a message plus related processing be M_{send} and M_{recv}, respectively. For GLOBAL, the overhead due to the load information exchanges is

$$C_{LIC} = \frac{(N-1) \times M_{recv} + M_{send}}{T} \times 100\%$$

for the LIC, and

$$C = \frac{M_{send} + M_{recv}}{T} \times 100\%$$

for the other hosts. Except for the LIC, the message overhead is *independent* of the system size N. In contrast, for the DISTED algorithm, the message overhead for *every* host is

$$C = \frac{(N-1) \times M_{recv} + M_{send}}{T} \times 100\%$$

because, during each interval of duration T, every host has to process the messages broadcast by every other host. † Compared to GLOBAL, we do not have a central point of failure and an extra level of indirection in the distribution of load information in DISTED, but the overhead is higher for every host, and grows linearly with the system size. Since the availability considerations are not important, as discussed above, the GLOBAL algorithm appears more attractive than DISTED.

One somewhat surprising result from Figure 5 is that the two drastically different algorithms, GLOBAL and LOWEST, provide almost identical response times under a wide range of system loads. The GLOBAL algorithm uses more extensive system state information in an effort to make optimal transfer decisions. To achieve this, load information is exchanged at a high frequency, thus incurring high overhead. In the simulation runs, the value of T that provides the best performance for GLOBAL is between 0.75 and 3 seconds. At such a high frequency, 1-3% of the CPU time in every host is spent exchanging load information. The LOWEST algorithm does

† Due to the policy of not sending out the local load information if it is the same as last time, the actual overheads of GLOBAL and DISTED are lower than presented here, typically by 40-70%, but the order analyses here are still valid.

not attempt to select the globally "best" host for job transfer, but rather only selects the least loaded among a small group of randomly picked hosts. Although the time it takes to poll the hosts directly increases the response time of the waiting job, more up-to-date load information is used for job placement. A main reason GLOBAL is not able to perform better than LOWEST is that there exists a fundamental contradiction between the need to frequently update the load vectors at each host and the low utilizations of the load vectors. If the exchange period is 1.0 second, and the rate at which transfer decisions are made by a host is one job every 10 seconds, then 90% of the load exchanges are wasted.

The above discussion seems to confirm the assertion made by Eager et al. [9] that more complicated algorithms than THRHLD and LOWEST are not likely to provide substantially better performance. In fact, the difference between LOWEST and the unrealistic NoCOST algorithm is already quite small. However, we do not observe the significant improvement from RANDOM to THRHLD, which was observed by Eager et al. This may be due to the fact that these authors did not consider the message exchange costs explicitly in their study.

4.2. Effects of System Scale

Scalability is an important issue in load balancing. On the one hand, a larger pool of hosts might improve the performance of load balancing. On the other hand, the overhead of load balancing might grow with system size, and the management of the system becomes harder. The average response times of the ten algorithms in systems containing 7, 14, 21, 28, 35, 42, and 49 hosts are shown in Figure 6. To make the comparisons meaningful, the overall system utilizations are selected to be within a narrow range (see the host utilization numbers for the NoLB case at the top of Figure 6), and the response times are normalized against that of NoLB.

The negative slopes of the NoCOST and PartCOST, as well as those of some of the other algorithms, suggest the presence of economies of scale. As the number of hosts in the system increases, the probability of finding a lightly loaded host increases, and the average response time can be expected to decrease. This is most obvious for the NoCOST case, where the overhead costs are not considered. For the more realistic algorithms, the overhead may increase with the system size, making the increase in system size a mixed blessing. Therefore, the scalability of an algorithm is an important property. On the other hand, it is interesting to observe that, as the number of hosts increases beyond 28, the response times improve very little. Therefore, a scalability up to a few tens, or at most a few hundreds of hosts, seems sufficient. Beyond that point, it makes more sense to implement load balancing using several clusters and perform inter-cluster load balancing using longer-term load information. This observation further enhances the values of algorithms such as CENTEX and GLOBAL.

Again, in Figure 6, we observe the relative performances of the algorithms. The scalability of RANDOM, THRHLD and LOWEST is very good. (Their curves are almost parallel to that of NoCOST.) This is because the number of hosts polled by the algorithms when a placement decision is made is independent of system size. Comparing GLOBAL and DISTED, we see that the former scales much better, which may be explained by the overhead analyses in Section 4.1. We can see two conflicting factors in action by looking at the performance of the DISTED algorithm. On the one hand, an increase in system size makes it easier to find a host with low load. On the other hand, the message overhead grows linearly with system size. The composite effect is a

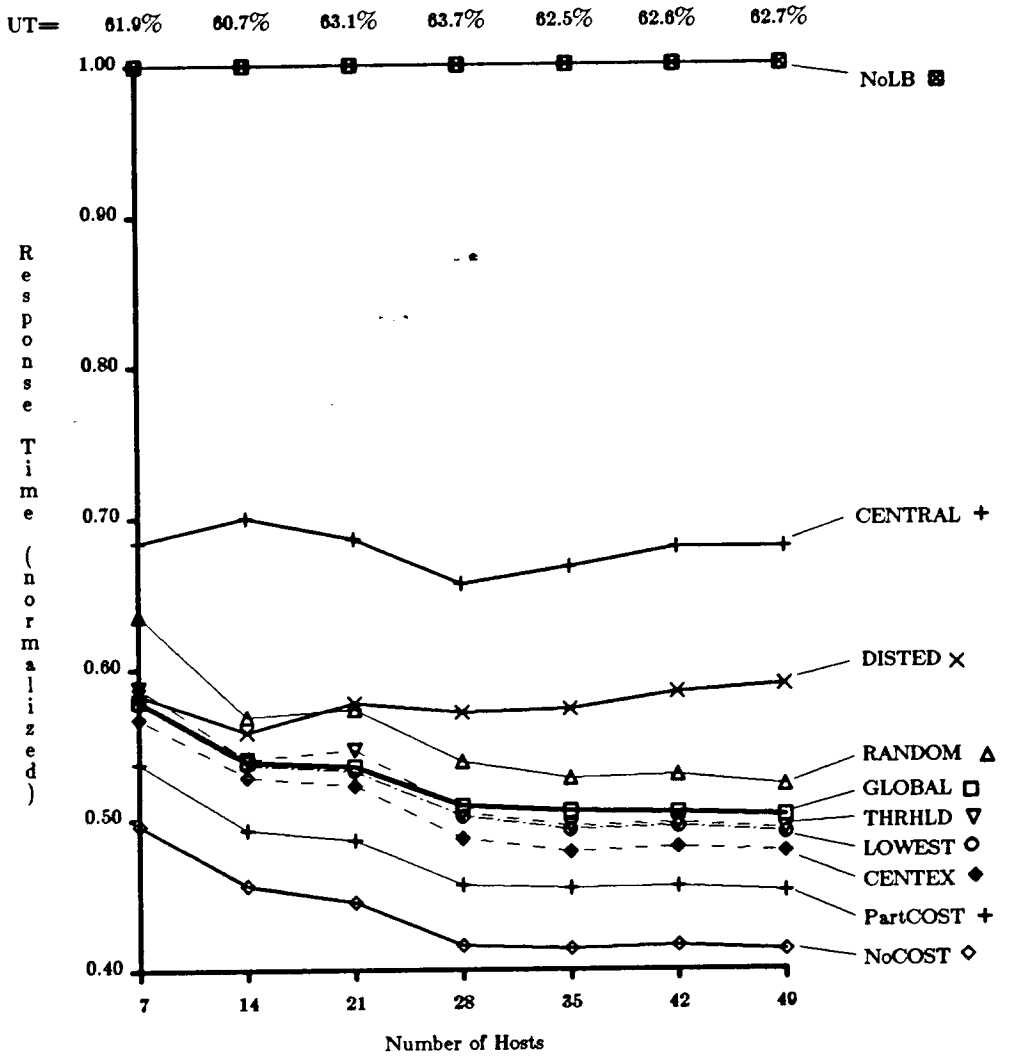


Figure 6. Average response time with different system sizes.
(normalized against the NoLB case)

moderately rising curve for the normalized response time. CENTRAL remains the worst in all cases, while CENTEX demonstrates very good performance and satisfactory scalability.

4.3. Effects of Load Balancing on Individual Hosts

In the previous studies of load balancing, it has been frequently assumed that the hosts in the system are subjected to the same level of load [9, 16, 20]. (The job arrival rates and the processing rates of all the hosts are the same.) However, this is usually not the case in production environments. It is very interesting to study the effect of load balancing on the individual hosts, especially those originally with light loads. At the beginning of this research, we conjectured that, while load balancing may improve overall system performance and that of the heavily loaded hosts, the lightly loaded ones may suffer degradations in their performance because additional jobs are transferred to them. We were, therefore, pleasantly surprised by the results from the simulations. The average response times of the individual hosts, with and without load balancing, are

shown in Figure 7. As can be observed, the performances of *all* hosts are generally improved, with the hosts under heavy loads showing greater improvements. Figure 7 clearly demonstrates the power of dynamic load balancing: system performance may be greatly improved by taking advantage of the temporal differences among the hosts' loads, and even hosts with light loads benefit as congestions on them, though infrequent, can be relieved by other hosts.

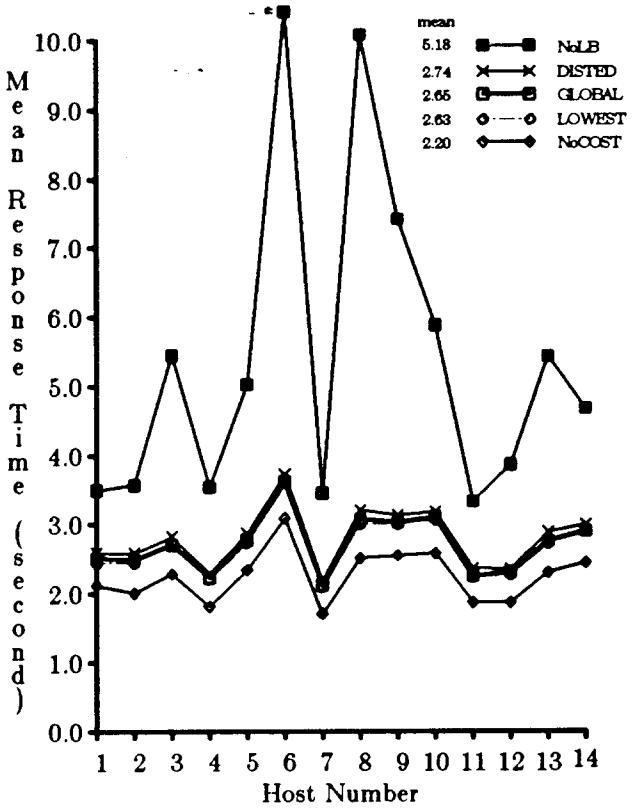


Figure 7. Mean response times of individual hosts.
(Utilization without load balancing is 63.3%.)

Another beneficial effect of load balancing revealed by Figure 7 is that it makes the response times more *predictable*. For some environments, this is even more important than the reduction in the mean response time. Figure 8 provides a direct measure of this effect: while the average response time is cut by a factor of 1.5 to 2.0, its standard deviation is cut by a factor of 2 to 4. The measurements in Figures 7 and 8 are taken when the system is moderately loaded (the utilization for the NoLB case is 63.3%). The improvements of the mean and standard deviation of response time are observed to be more drastic when the system load level is higher.

The term load balancing has in it the implicit meaning of equalizing the loads of the participating hosts. Though this is not our objective, the equalizing effect of the algorithms studied in this paper can be clearly seen in Figures 9 and 10. This is more pronounced with GLOBAL and DISTED than with LOWEST because of the attempt of the former two algorithms at system-wide optimization. It is interesting to note, in Figures 7 and 10, that the average response time of all the jobs originating from a host may decrease while the host's utilization increases. The amount of overhead introduced by a load balancing algorithm can be found in Figure 10 by subtracting

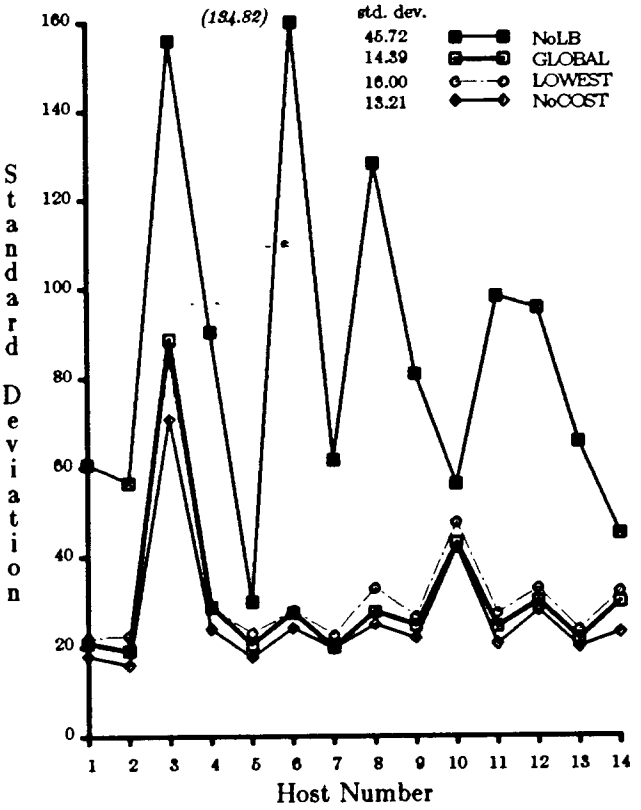


Figure 8. Standard deviations of response times of individual hosts.
(Utilization without load balancing is 58%.)

the average CPU utilization with no load balancing from that obtained with a specific algorithm. As can be expected, the overhead of GLOBAL is higher than that of LOWEST, while both provide similar performance. This is mainly because of the high frequency of load information exchanges in GLOBAL. The overhead on the LIC in GLOBAL is proportional to the system scale, and is higher than that on the other hosts. This is reflected by the significant increase in utilization of host 1, which we use as the LIC. While this reveals the limitation of the scaling ability of the algorithm, we do not consider it a serious drawback because our simulation results show that a single LIC is capable of supporting 50-100 hosts under our cost assumptions. At that point, the economies of scale have almost no effect, and it is reasonable to implement load balancing in several clusters.

4.4. Parameter Selection and Adaptive Load Balancing

Once the load balancing algorithm is decided, the performance is still sensitive to the specific parameter values adopted. In this section, we assess the degree of such sensitivity. The adjustable parameters depend on the algorithm. For all of the algorithms, we have a local load threshold and a job threshold. In addition, for the periodic information policies, we have the load exchange period, whereas for the non-periodic policies, we have the probe limit. Figure 11 shows the performance of the GLOBAL algorithm under various parameter combinations. The local load threshold is set to zero. We can see that the exchange period has a strong influence on the

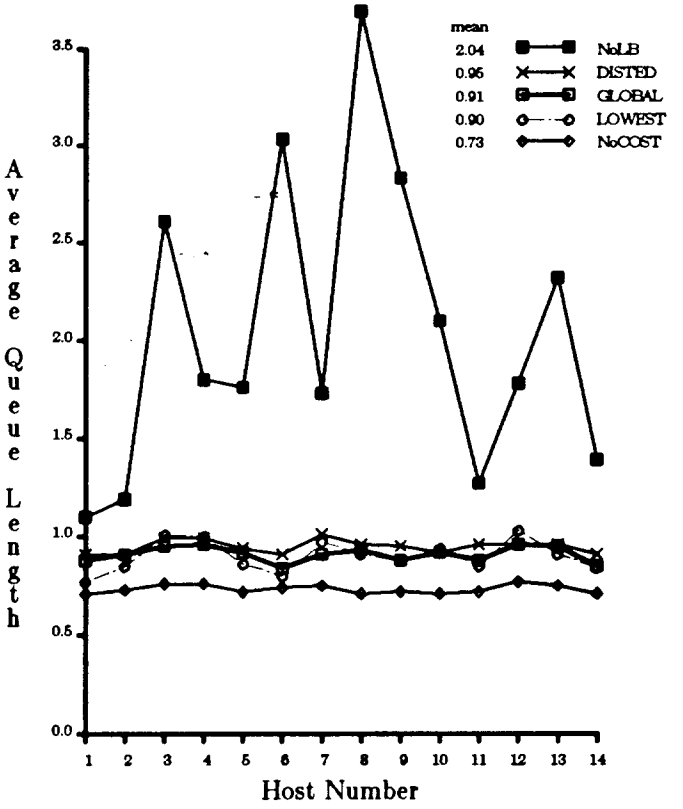


Figure 9. Average queue lengths of individual hosts under different load balancing algorithms.

performance. When the period is too short (e.g., 0.35 second), the overhead is so high that, even though the load information on which the transfer decisions are based is very up to date, the performance suffers. On the other hand, if the exchange period is too long (e.g., 10 seconds), the load information is so out of date that frequent mistakes are made in job placements. (Jobs are sent to hosts with equal or higher load than the local host.)

In contrast, performance seems to be less sensitive to the job threshold. (The average response time when only jobs with CPU execution time greater than 1.0 second are considered for load balancing is close to those when jobs above 0.5 or 2.0 seconds are considered.) This observation supports our earlier claim that only an approximate separation between big and small jobs is necessary to achieve good performance. Looking more closely, we again observe a similar pattern with the job threshold: when the job threshold is too high (e.g., 3 seconds), the full potential of load balancing is not realized, whereas when it is too low (e.g., 0.25 second), the overhead of job transfers outweighs the benefit, and performance becomes worse. There is also interaction between the two parameters; when the exchange period is lengthened, the corresponding optimal job threshold increases.

It is important to recognize that the combination of parameters that yields the best performance is highly dependent on the system load level. Table 1 attempts to illustrate this. Generally speaking, the higher the load, the higher the job threshold and the longer the exchange period should be. For LOWEST, an increase in the probing limit may yield poorer performance

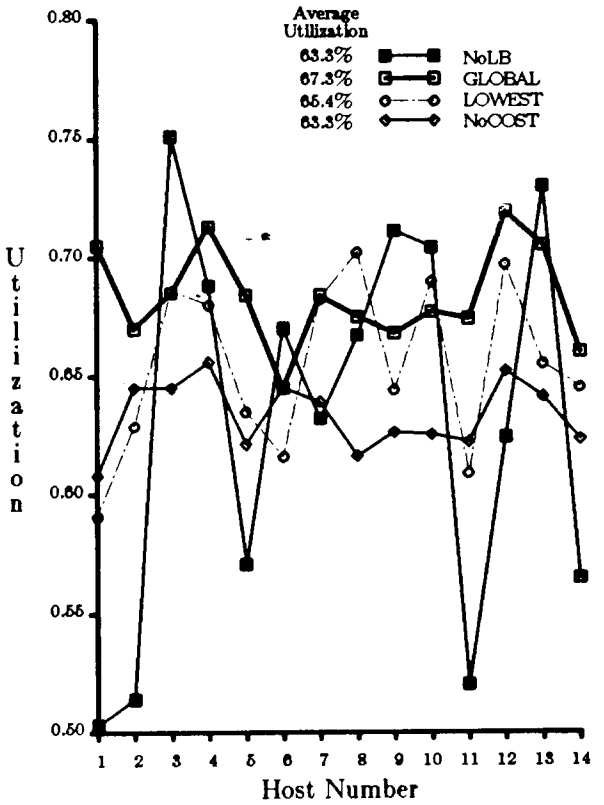


Figure 10. Utilizations of individual hosts under different load balancing algorithms.

when the load is high.

Table 1. Optimal Parameter Values under Different System Load Levels (28 hosts).

Utilization (%)	48.1	56.2	63.3	72.3	79.1	85.1	92.1
GLOBAL							
Load Exch Pd	0.5	0.5	0.75	1.0	1.5	2.0	3.0
Job Thrd	0.4	0.5	0.5	0.75	1.0	1.5	2.0
DISTED							
Load Exch Pd	1.5	2.0	2.5	3.0	4.0	5.0	7.5
Job Thrd	0.4	0.5	0.75	1.0	1.5	2.0	2.5
LOWEST							
Job Thrd	0.4	0.5	1.0	1.0	1.5	2.5	4.0
Load Thrd	0	0	0	0	0	0	0

(The numbers are approximate, as only a sparsely allocated set of operating points in the multi-dimensional parameter space are tested for each algorithm.)

The sensitivity of load balancing performance over the parameter values suggests that some form of *adaptive* load balancing may be able to provide good performance when system load changes widely. Under adaptive load balancing, the system load is constantly monitored, and changes in algorithms and/or adjustable parameters are made as the load changes so that the system is always operating at, or close to, the optimal point. Supporting multiple algorithms involves complicated implementation, and changes in algorithms cannot be made frequently. Furthermore, for the most promising algorithms, GLOBAL, CENTEX, and LOWEST, the performance differentials are quite small. Consequently, the gain from switching algorithms is probably insignificant, and therefore not worth the effort. However, we are not making a general statement here; for environments different from ours, and for algorithms other than the ones we studied, using different algorithms under different loads might be quite advantageous.

In contrast to algorithmic change, parameter adjustments are much simpler, and capable of significantly improving performance when the system load fluctuates widely. Here, we need a system-wide mechanism that monitors load conditions and makes adjustment decisions. GLOBAL and CENTEX are the most appropriate for this purpose. The LIC periodically receives load information from all the hosts, and can use such information to deduce the system state. It can then send to the hosts the parameter values they should use.

4.5. Avoidance of Instability

The problem of instability introduced by load balancing is of major concern to the researchers in this field. It is feared that, because of the delay in load information exchanges, several hosts may transfer jobs to a once lightly loaded host, and cause it to be overloaded. After the load information is updated, some other host(s) may become the victim(s). We call such phenomenon of overloading hosts in turn *host overloading*. Another form of instability is *job thrashing*, in which jobs are transferred too many times (or even for an indefinite number of times, as analytically shown in [9] for RANDOM) in an attempt to find the optimal host for job execution. Host overloading causes performance degradation because of unstable and uneven load distribution among the hosts, whereas, for job thrashing, degradation is mainly due to excessive job transfer overhead. Since we are mostly concerned with algorithms that transfer jobs only once, we will study the host overloading problem here.

We consider a job transfer *wrong* if the destination host's CPU queue length is equal to or greater than that of the originating host. There is a distinction between transferring a job wrongly and collectively overloading a host; the former by itself will only increase the particular job's response time, whereas the latter will potentially cause system-wide performance degradation, due to the aggravated effects of the individual wrong transfers. This problem can be serious because usually the transferred jobs are big. In order to quantitatively measure the level of host overloading occurring in a system, we define the *host overloading factor* τ to be the percentage of wrong job transfers over all transfers:

$$\tau = \frac{number\ of\ wrong\ transfers}{total\ number\ of\ transfers} \times 100\%$$

There are a number of factors that affect the level of host overloading, all having something to do with the rate at which wrong transfers are made to a host because τ is roughly proportional

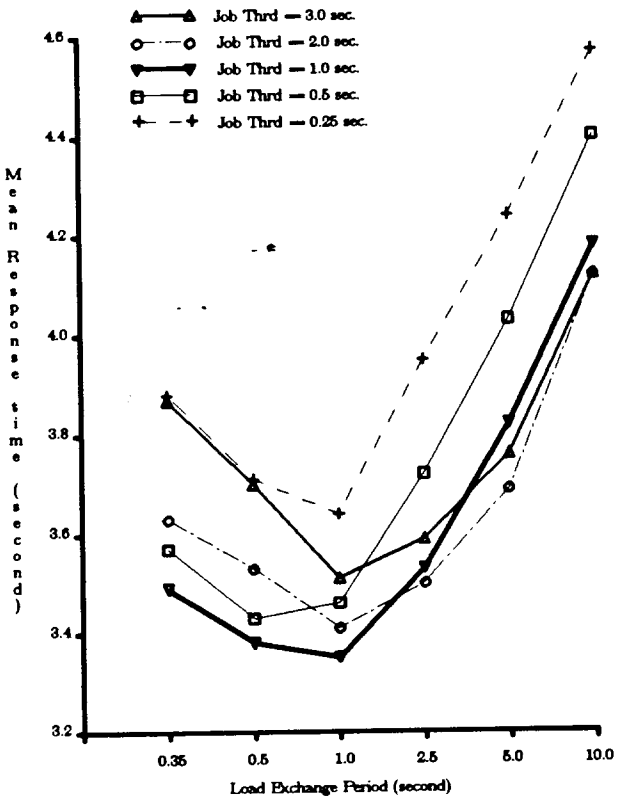


Figure 11. Effect of Adjustable Parameters on Load Balancing Performance.
(GLOBAL, 14 hosts, utilization without load balancing: 72.3%)

to this rate. First, the staleness of load information has a deciding effect. The staler the information, the more the jobs that are transferred wrongly. Therefore, the non-periodic information policies that collect load information on demand are less susceptible to host overloading than the periodic policies. Another important factor is the rate at which jobs that are candidates for transfer arrive. This depends on the system load level and the job threshold. The higher the load and the lower the job threshold, the larger the percentage of eligible jobs. To verify our intuitive argument, we calculated T in simulation experiments for the GLOBAL algorithm using various load exchange periods and job thresholds. Since it is difficult to consider three factors all changing at the same time, we fixed the system load level at 79%. Such a system-wide utilization is high, and host overloading may be expected to be quite serious. The results are shown in Figure 12, and agree with our intuition. It seems that host overloading does not have as disastrous effects on system performance as we feared: very good performance can be achieved even when there exists light overloading ($T < 10\%$).

Besides load update frequency and job threshold, the system scale also affects host overloading, but to a lesser degree. It is important to know the number of hosts with the least load. For the algorithms studied in this paper, placement decisions are based on the instantaneous CPU queue lengths of the hosts. Since there may be more than one host with the same shortest queue length, the transferred workload may be shared by them, thus reducing overloading. A larger system size makes such situation more probable. On the other hand, in a larger system, there are

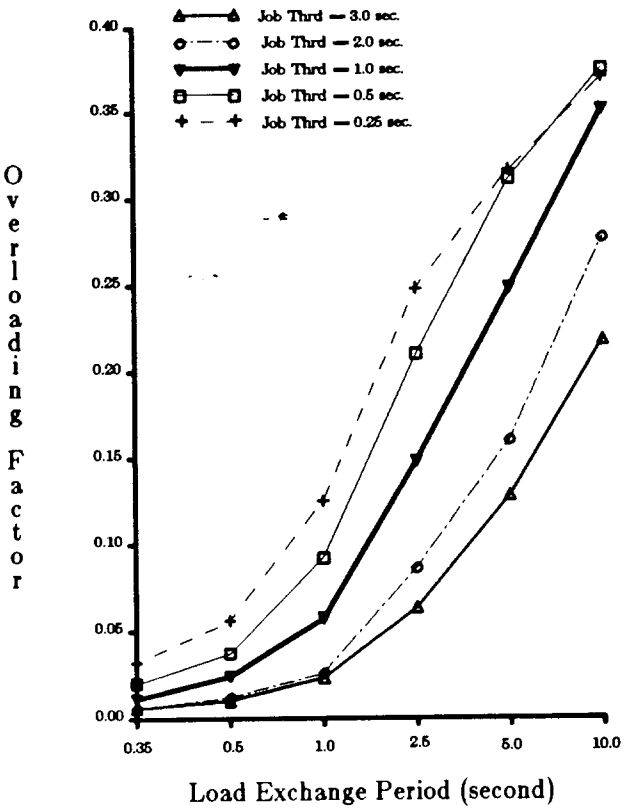


Figure 12. Percentage of wrong job placements for GLOBAL under various load exchange periods and job thresholds. (Number of hosts: 14, Average utilization: 79%)

also more sources of transferred jobs. To quantitatively study the number of hosts with the least load as a function of system size and load update period, we recorded the load vector at a high frequency during a simulation experiment for GLOBAL, and counted the number of hosts with the least number of jobs at their CPU's. The actual shortest queue length is unimportant because we are only concerned with the relative distribution here. Figure 13 shows the distributions for systems with 14 and 28 hosts, and the exchange period fixed at 5 seconds. For shorter exchange periods, the means of the number of hosts with the least load are slightly lower. We find that the probability of having only one or two hosts with the least load is non-negligible; hence host overloading can occur, as revealed using another metric in Figure 12. Consider the following case, which was found to be typical: for a system with 14 hosts and a load level of 80%, the total rate at which jobs are transferred by the GLOBAL algorithm using a job threshold of 1.0 second is 1-2 jobs/second. This means that, if we update load information every 5 seconds, 5-10 jobs may be transferred to the single host that used to have the least load! This range is reduced to 1-2 jobs if the exchange period is 1.0 second, and even lower if the system load is not at such a high level. Hence, we see that whether host overloading occurs depends primarily on the system load and the load exchange period.

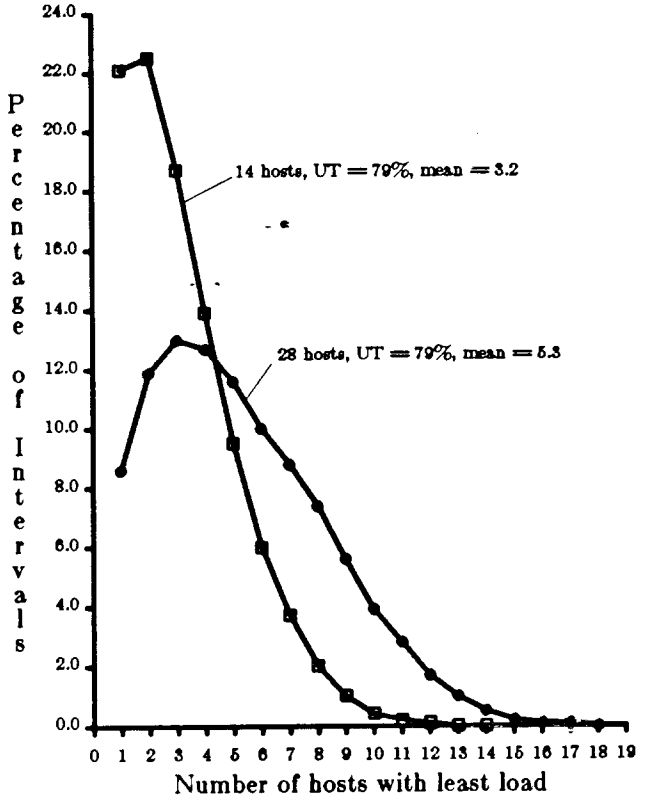


Figure 13. Distribution of the number of hosts with the least load.
Load Expd = 5.0 sec., Job Thrd = 1.0 sec.

4.6. Immobile Jobs

Throughout our studies so far, we have assumed that the jobs are *mobile*, that is, they can be executed on any host in the system with exactly the same results. Although this assumption holds for a large subset of the jobs, we do observe that some of the jobs are immobile. Examples include jobs that perform local services and/or require local resources, such as system daemons, login sessions, mail and message handling programs, and so on. There are also highly interactive jobs, such as command interpreters and editors, for which remote execution will result in poor performance due to network latencies. Any implementation of load balancing must take the effects of these immobile jobs into consideration. We define the *immobility factor* to be the percentage of jobs that have to be executed on the local host, but are otherwise eligible for load balancing. By varying the value of the immobility factor, the effect of immobile jobs is revealed. For a system of 28 hosts with an average CPU utilization of 63.7%, the results are shown in Figure 14.

The concave shapes of the curves are encouraging, as they indicate that effective load balancing is still possible even if a significant proportion of the jobs are immobile. For an immobility factor of 0.4, the mean response time is only slightly higher than that for the case in which all jobs are mobile (the immobility factor being 0). This observation is not surprising because load balancing is achieved by only transferring a portion of the eligible jobs. (Typically, 50-70% of the eligible jobs were actually transferred in the simulation experiments.) Consequently, even though some of the eligible jobs are immobile, the rest of them can still produce most of the

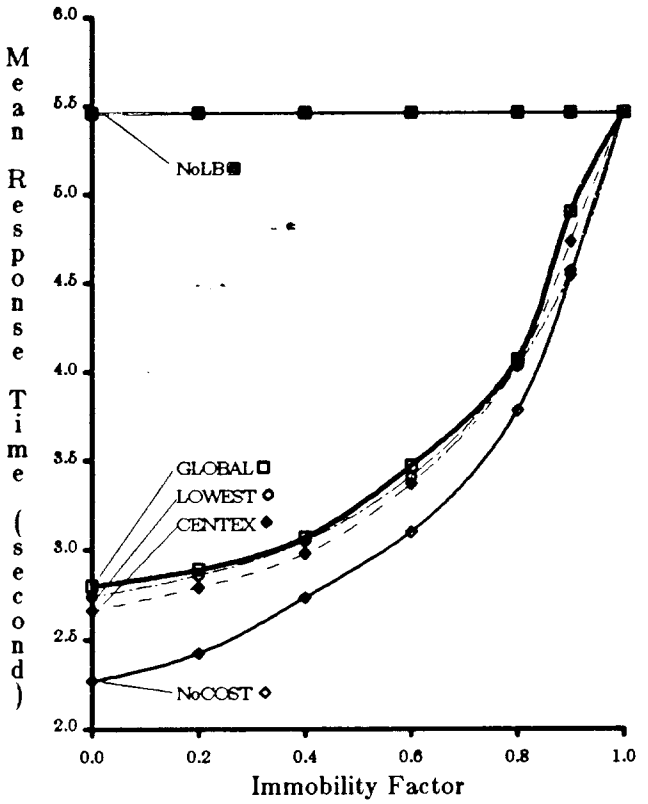


Figure 14. Effect of immobile jobs.

performance benefits due to the balancing effect.

5. Conclusions

In this paper, we studied the load balancing problem using simulation models driven by job traces collected from a production system. We simulated a CPU scheduling policy that is believed to be representative, and we considered explicitly the costs of load information exchange and job transfers. Because of the generality of the model and the use of live system data, the results of our simulation are believed to be more reliable than those from analytic models or simulations driven by probability distributions. On the other hand, our results might be biased towards a particular type of computing environment.

Seven load balancing algorithms were studied, including both ones using periodic information policies and ones using non-periodic policies. We found that load balancing using any reasonable algorithm can provide substantial performance improvement over the NoLB case. Specifically, the average response time of all the jobs may be reduced by 30-60%, and the reduction in its standard deviation is even more drastic, making the job response times much more predictable than in the NoLB case. The higher the load, the greater the improvement, and longer jobs benefit more from load balancing. Looking more closely, we found that the performance of all hosts, even those originally with light loads, improve under effective load balancing. This is somewhat counter intuitive, but very encouraging: by cooperating with each other, no one loses. We also observed a strong tendency of load balancing to equalize the loads of the individual hosts; both

the utilizations and the average CPU queue lengths of the hosts cluster within a small range.

By varying the size of the system, we observed significant but limited economies of scale. For example, when four systems each with 7 hosts, or two systems each with 14 hosts, are combined into a single system of 28 hosts, the average response time is significantly reduced for the algorithms with good scalability. However, beyond 28 hosts, the improvement diminishes quickly. Consequently, a scalability of an algorithm up to 50-100 hosts seems to be sufficient, and clustering techniques should be used in large scale systems to avoid the potentially increasing overhead and the management complexities.

For the periodic load information policies, we found that the global approach has much less overhead than the distributed approach, and, therefore, performs better and is more scalable. The periodic and non-periodic policies provide comparable performances under our cost assumptions. The algorithms that collect load information on demand (RANDOM, THRHLD and LOWEST) have the advantages of lower message exchange overhead, of being able to scale better, and of being less susceptible to host overloading. On the other hand, the algorithms that rely on periodic load exchanges (GLOBAL, CENTEX, and DISTED) have the advantages of being able to potentially choose the optimal hosts for job transfers, thus offering better performance, and of not subjecting the jobs eligible for transfer to the delays in getting load information.

The performance of load balancing using the algorithms studied in this paper is found to be quite sensitive to the values of the local load threshold, load exchange period, and host probing limit. The combinations of the parameter values that provide optimal performances are in turn dependent on the system load level. Consequently, adaptive load balancing has the promising potential of maintaining optimal performance under changing system configuration and load. The GLOBAL and CENTEX algorithms are the most appropriate for this because of the presence of the LIC. More research is called for in this area.

Host overloading is not significant for non-periodic algorithms, but may be serious with the periodic algorithms. The deciding factors are the load update frequency, the system load, the job threshold, and the system size. By using reasonably up-to-date load information and only transferring a small percentage of jobs, host overloading can be effectively alleviated. Suboptimal placement decisions may produce better performance than "optimal" decisions, because overloading on one or a small number of hosts may thus be avoided. Host overloading is not as serious as we expected — very good performance can be achieved even when it occurs occasionally.

The impact of immobile jobs on load balancing is found to be less serious than the immobility factor might suggest: most of the performance gains are still retained even when up to 50% of the jobs are immobile.

We have been very much encouraged by the trace-driven simulation approach taken in this research; it proves to be capable of handling greater complexities and of providing more credible performance results than the approaches used before in load balancing research. On the other hand, we only used data from a particular type of time-sharing environment, and so the generality of our results is limited. The simplifying assumptions made in this research, though less unrealistic than those of the previous studies, may also have introduced errors in our results. It would be very interesting to apply the techniques used in this research to other types of computing environments, especially server-based workstation environments, and to compare the findings. Such

efforts are currently being planned.

In view of the proliferation of distributed systems, and of the great potential of load balancing as demonstrated in this research and by other authors, it is highly desirable that load balancing be made a standard service in future distributed systems to substantially increase the performance of the system without adding any resources. Unfortunately, only a few implementations exist, and most of them were done in an *ad hoc* fashion [2, 12, 13, 19]. Besides the implementation difficulties involved, a general lack of understanding of the performance characteristics of the algorithms proposed and the engineering tradeoffs involved are the major obstacles. Trace-driven simulation appears to be an appropriate tool for load balancing studies, and should be well exploited before an implementation effort starts, because the latter is much more costly.

6. Acknowledgements

The author wishes to express his deep gratitude to Domenico Ferrari for his invaluable advice and continued support and encouragement throughout the course of this research. Comments by David Anderson, Hamid Bahadori, Luis Felipe Cabrera, Joseph Pasquale, and Harry Rubin inspired a significant revision and improvement to the paper, and are gratefully acknowledged.

7. References

[1] A. Barak and A. Shiloh, "A Distributed Load Balancing Policy for a Multicomputer," Department of Comp. Sci., The Hebrew University of Jerusalem, 1984.

[2] B. Bershad, "Load Balancing with Maitre d'," Tech Report, UCB/CSD 85/276, Computer Science Division, University of California, Berkeley, December 1985.

[3] S. H. Bokhari, "Dual Processor Scheduling with Dynamic Reassignment," IEEE Trans. Soft. Eng., SE-5,4, July 1979, pp. 341-349. pp. 47-55.

[4] R. Bryant and R. Finkel, "A Stable Distributed Scheduling Algorithm," Proc. International Conf. on Distributed Processing Systems, 1981, pp. 314-323.

[5] L. F. Cabrera, E. Hunter, M. Karels, and D. Mosher, "A User-Process Oriented Performance Study of Ethernet Networking under Berkeley UNIX 4.2 BSD," To appear in IEEE Trans. Soft. Eng., also as Tech. Report, UCB/CSD 84/216, Computer Science Division, University of California, Berkeley, December 1984.

[6] L. F. Cabrera, "The Influence of Workload on Load Balancing Strategies," Proc. 1986 Summer USENIX Conference, Atlanta, GA, June 1986, pp. 446-458.

[7] Y. Chow and W. Kohler, "Models of Dynamic Load Balancing in a Heterogeneous Multiple Processor System," IEEE Trans. Comp. C-28,5, May 1979, pp. 354-361.

[8] D. Eager, E. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Dynamic Load Sharing," Tech Report 85-04-01, Dept. of Comp. Sci, Univ. of Washington, April 1985.

[9] D. Eager, E. Lazowska, and J. Zahorjan, "Dynamic Load Sharing in Homogeneous Distributed Systems," IEEE Trans. Soft. Eng., SE-12,5, May 1986, pp. 662-675.

[10] D. Ferrari, "A Study of Load Indices for Load Balancing Schemes," Tech Report, UCB/CSD 85/262, Computer Science Division, University of California, Berkeley, October 1985; also in: G. Serazzi, Ed., "Workload Characterization of Computer Systems and Computer Networks," North-Holland, Amsterdam, 1986.

[11] A. Hac, and T. J. Johnson, "A Study of Dynamic Load Balancing in a Distributed System", Proc. ACM SIGCOMM Symposium on Communications, Architectures and Protocols, Stowe, Vermont, August 1986, pp. 348-356.

[12] R. Hagmann, "Process Server: Sharing Processing Power in a Workstation Environment," Proc. Principles of Distributed Computing, Cambridge, MA, May 1986.

[13] K. Hwang, W. Croft, G. Goble, B. Wah, F. Briggs, W. Simmons, and C. Coates, "A UNIX-based Local Computer Network with Load Balancing," IEEE Computer, 15,4, April 1982, pp. 55-66.

[14] E. Lazowska, J. Zahorjan, D. Cheriton, and W. Zwaenepoel, "File Access Performance of Diskless Workstations," Tech Report 84-06-01, Dept. of Comp. Sci, Univ. of Washington, June 1984.

[15] W. Leland and T. Ott, "Load-balancing Heuristics and Process Behavior," ACM SIGMETRICS Conf., May 1986, pp. 54-69.

[16] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," Proc. ACM Computer Network Performance Symposium, April 1982,

[17] H. S. Stone, "Multiprocessor Scheduling with the Aid of of Network Flow Algorithms", IEEE Trans. Soft. Eng., SE-3,1, January 1977, pp. 85-93.

[18] H. S. Stone, "Critical Load Factors in Two Processor Distributed Systems", IEEE Trans. Soft. Eng., SE-4,3, May 1978, pp. 254-258.

[19] M. Theimer, K. Lantz, and D. Cheriton, "Preemptive Remote Execution Facilities for the V-System," Tech Report No. STAN-CS-85-1087, Computer Science Dept., Stanford Univ., September 1985.

[20] Y. Wang and R Morris, "Load Balancing in Distributed Systems," IEEE Trans. Comp. C-34,3, March 1985, pp. 204-217.

[21] S. Wu and M. Liu, "Assignment of Tasks and Resources for Distributed Processing," Proc. COMPCON, Fall 1980, pp. 655-662.

[22] S. Zhou, "An Experimental Assessment of Resource Queue Length as Load Indices," Tech Report, UCB/CSD 86/298, Computer Science Division, University of California, Berkeley, April 1986.