Productivity Engineering in the UNIX† Environment

An Empirical Investigation of Load Indices for Load Balancing Applications

Technical Report

S. L. Graham
Principal Investigator

(415) 642-2059

"The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government."

Contract No. N00039-84-C-0089

August 7, 1984 - August 6, 1987

Arpa Order No. 4871

†UNIX is a trademark of AT&T Bell Laboratories

An Empirical Investigation of Load Indices for Load Balancing Applications†

Domenico Ferrari and Songnian Zhou

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Abstract

In this paper, we empirically evaluate the quality of several load indices in the context of dynamic load balancing. We have implemented a load balancer for Sun/UNIX‡ environments. In our experimental setup, six Sun-2 workstations were driven by job scripts, and job response times were measured while loads were being balanced and various load indices used to make job placement decisions. We study the effects on performance of the choice of load index, the averaging interval, the load information exchange period, and the characteristics of the workload. Measurements show that the performance benefits of load balancing are indeed strongly dependent upon the load index. Load indices based on resource queue lengths are found to perform better than those based on resource utilization, and the use of an exponential smoothing method yields further improvement over that of instantaneous queue lengths.

† 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 authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Research Projects Agency or of the US Government.

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

1. INTRODUCTION

In a loosely-coupled distributed system, the potential for resource sharing and its possible rewards are substantial. Two frequently cited advantages of resource sharing are the larger number of accessible resources, in terms of both type and quantity, and the higher reliability that may result from the multiplicity of available resources. In order to share these resources effectively, some measure of the loads being imposed on the resources has to be made available to the clients. The information about resource loads is part of the system's state, and is among the most rapidly changing aspects of it. Since the loads are likely to be changing all the time, load information tends to become stale rapidly. To quantify the concept of load, we use a *load index*, which preferably is a non-negative variable taking on a zero value if the resource is idle, and increasing positives values as the load increases. This paper is concerned with the quality of the possible load indices for hosts in a particular but important application of load indices, that of load balancing in distributed systems.

A job arriving at a host will very likely demand services from a number of resources (e.g., CPU and disks). Hence, it is important to define not only the load on a single resource in a host, but also that of the host viewed as a collection of resources. Since the resource consumption patterns of the jobs are likely to be different, it may not be meaningful to talk about "the load" of the host. For example, the CPU may be heavily congested, while the disks are not. In this case, to an incoming CPU-bound job the host's load is very high, whereas to an incoming I/O-bound job the host's load is low because it will not experience much queueing at the disks. This observation is formalized in [Ferrari86], where a job type-dependent load index based on the resource queue lengths is proposed and experimentally evaluated.

Load information is important since it can serve as the basis of the efforts to improve the system's performance by redistributing the loads. It is frequently observed that, in a distributed system, the loads of the hosts are not evenly distributed all the time. Livny and Melman pointed out that, for a queueing system consisting of multiple homogeneous service centers with Poisson arrivals of identical rates, the probability of some hosts being idle while some others have more than one job can be very significant; hence, redistributing the workload among the resources has the potential of improving performance [Livny82].

In order to evaluate the quality of a load index for load balancing, we specify a number of criteria, or desirable properties. These criteria, in turn, are dependent on the objective of load balancing, i.e., the *performance index* that is to be optimized by balancing the loads. In this research, we are mostly concerned with interactive computing environments, where the job response time and its predictability are very important measures of system performance. Therefore, we use the mean job response time as our performance index, supplemented by the standard deviation of the response times. A good load index should:

1) be able to reflect our qualitative estimates of the current load on a host;

2) be usable to predict the load in the near future, since the response time of a job will be affected more by the future load than by the present load;

3) be relatively stable; i.e., high-frequency fluctuations in the load should be discounted, or ignored;

4) have a simple (ideally, linear) relationship with the performance index, so that its value can be easily translated into that of the expected performance.

We recognize that it is difficult, perhaps impossible, to find a load index that satisfies all of the above requirements, as they may even turn out to be contradictory. But a load index may be judged by the degree to which it meets the above criteria.

A number of load indices have been proposed in the past, most of them related to the load balancing problem. In this paper, we conduct an empirical comparison study of a number of those indices, and attempt to rationalize and generalize our observations. The experimental environment we use includes a load balancer running on diskless Sun/UNIX workstations. By driving the workstations with job scripts and by using different load indices, we were able to measure the mean response times and compare them. In the next section, we examine the types of load index proposed and used in the past. The load balancer and the workloads we used are described in Section 3. In Section 4, we discuss the design of the experiments and their results. The major results are summarized in Section 5.

2. LOAD INDICES

A wide variety of load indices have been explicitly or implicitly used in the literature, mostly in load balancing schemes. For example, in most of the studies using queueing network analysis, as well as some in other studies, the CPU queue length was used as the load index [Chow79, Eager86a, Eager86b, Lee86, Livny82, Wang85, Zhou86]. Some other authors used the CPU utilization [Alonso86, Ezzat86]. Other possibilities include the normalized response time [Hwang82] (defined as the ratio between the response time of a process on a loaded machine and its response time on the same machine when it is empty), the remaining processing time of all the jobs running on a host, the processing time accumulated by the active processes [Hac87], and the total processing time of the active processes [Leland86]. Functions of the above simple variables have also been used [Ezzat86, Ferrari86, Zhou87a]. In a number of studies in which reducing job response time was the objective of load balancing, the estimated response time was used as the load index [Bryant81, Carey85]. Performance improvements were often reported using the indices discussed above. However, since no systematic and comprehensive comparisons between the indices have been made, their relative merits remain unclear. We note with regret that, in most cases, the authors did not even provide a scientific justification for the choice of the load index.

The first systematic attempt to study the load indices to be used in load balancing was made in [Ferrari86]. Based on mean value analysis, a linear combination of resource queue lengths was proposed as a load index. In that linear combination, the coefficient of a resource queue length is the amount of service time that the particular job being considered requires from that resource. Thus, if an incoming job requires s_j seconds of service from resource r_j, and the queue length of resource r_j is q_j, then the load index li of the host, *as perceived by this job*, is

$$li = \sum_{j=1}^{N} s_j \times q_j$$

where N is the total number of resources for which there is queueing in the host. This index was evaluated with measurement experiments under a production time-sharing workload [Zhou87b].

The index introduced in [Ferrari86] is response time oriented, and job dependent. Instead of a unique value at a particular moment in time, the load of a host differs for different jobs because of their varying resource demands, which are assumed to be known upon job arrival. This assumption enables us to predict the response time of a job more accurately, hence to make better load balancing decision. However, while we have found some simple relationships between the arguments of a job and the job's resource demands [Zhou87c], the assumption that the demands of a job are known in advance may be too strong in many cases. In this study, we investigate versions of the same load index in which the coefficients of the resource queue lengths are *job independent*, and only reflect the relative importance of the resources (with respect to a "basket" of jobs). For example, we can use unity as the coefficients to reduce the linear combination to the sum of the resource queue lengths, that is, in queueing modeling terms, "the number of jobs (or processes) in the system."

Our extensive measurements of production time-sharing workload show that the system load is changing quite rapidly [Zhou87b]. On top of a low-frequency main component, there are a number of high-frequency load components that may be regarded as "noise" rather than useful information. Using the instantaneous resource queue lengths may give excessive importance to such noise and lead to bad job transfer decisions. We used a smoothing algorithm to compute the time-averaged queue length and compared load balancing performance using smoothed queue lengths to that of the same scheme using instantaneous queue lengths.

3. SYSTEM AND WORKLOAD

In this section, we describe the experimental environment in which the measurements were taken, and the workloads used to drive the system.

System

We implemented a dynamic load balancer for Sun/UNIX environments. The structure of the system is shown in Figure 1†. The UNIX user interface program, *csh*, is modified so that the commands typed in by the user are intercepted, and some of them are transferred to some remote host for execution when the local host is heavily loaded‡. At startup time, the C-shell reads in a configuration file that specifies a list of job types

† To distinguish our modified C shell from the standard one [Joy80], we call it *C-shell*. The *R-shell*, to be described below, shares the same software with the C-shell, but its only function is to receive remote jobs and execute them.

‡ Our system is based on a modified C shell implemented at Berkeley by Harry Rubin and Venkat Rangan for the Berkeley UNIX 4.3 BSD system running on VAX machines [Joy83, McKusick85].

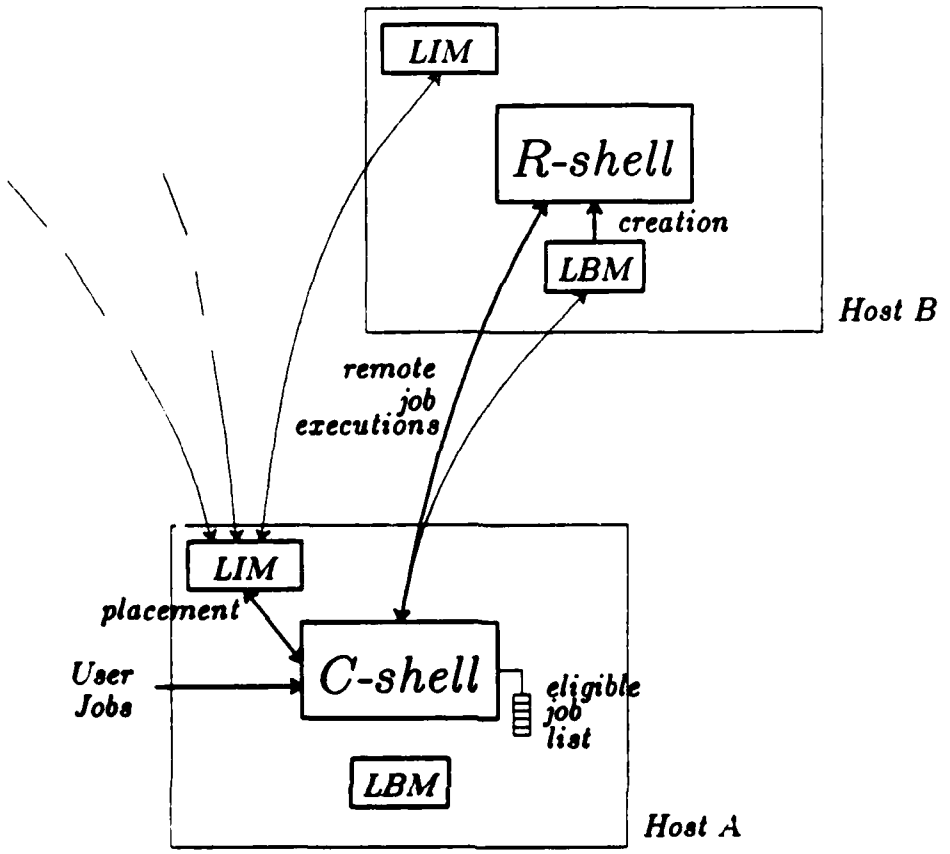

Figure 1. Structure of the load balancer.

that are eligible for remote execution*. When an eligible job is submitted by the user to the C-shell, the C-shell contacts the local *Load Information Manager* (LIM), a software module that constantly exchanges load information with its peers on other hosts and performs job placements. If the local host is heavily loaded, while some other hosts are not, one of the remote hosts is selected as the destination for the job. In any case, the placement decision is returned to the C-shell.

For remote execution, the C-shell contacts the *Load Balancing Manager* (LBM) on the destination host, which starts up an R-shell and establishes a stream connection between it and the home C-shell. The command line is transmitted over this connection to the R-shell after the user's identity has been authenticated, and an appropriate user environment set up there. Access control to files and other resources in the system is automatically enforced as the R-shell assumes the same user identity as that of the home C-shell. Since starting an R-shell is an expensive operation (several seconds of real time), we keep such a shell alive after the execution of the first job so that, if a later command from the same user login session is placed on the same host, we do not have to go through the same process described above. The R-shells on remote hosts act as agents for the

* This list is part of the context of each user, just like command aliases, and may be dynamically modified by the user to suit his or her needs.

home C-shell, and are terminated when the home C-shell exits. This scheme has the potential problem of R-shell proliferation. However, the code segments of all C-shells and R-shells on each host are shared, so that, when an R-shell is not active, almost no resources are consumed. Since files are retrieved from file servers, as the workstations are diskless, only the command line needs to be shipped, and the cost of file access is essentially the same from all hosts.

Load balancing algorithms have a strong influence on performance. We implemented and studied a number of algorithms using different methods for load information exchange and job placement [Zhou87a]. For this study of load indices, however, we just selected one of the best realizable algorithms, that is, the one we called GLOBAL. For every time period P, the LIM on each host extracts load information from the local kernel to compute the local host's load index. If the new value of the load index is significantly different from the previous one, the new value is sent to the *master* LIM, which collects load information from every host and broadcasts the entire load vector in each period P. When a job whose name is on the eligibility list is submitted to a host, the local LIM is contacted for job placement. If the local load is high, the host perceived by the local LIM to have the least load is selected, and the job is sent there.

The implementation described above provides a transparent, low-cost, and general-purpose load balancer whose installation requires no changes to the kernel† or to the application programs. Since the emphasis of this paper is on the measurement experiments we performed on the system, we will not describe the design and implementation issues in more detail. The interested reader is referred to [Zhou87a].

Workload

Workload characterization and selection are crucial to a measurement study. Although artificial workloads considerably increase the repeatability of experiments, they ought to represent natural workloads reasonably well, so as to strengthen our confidence in the results. We traced a production VAX-11/780 machine running under the Berkeley UNIX 4.3BSD system [Joy83, McKusick85] for an extended period of several months, and analyzed the types and frequencies of the commands executed by the system. On the basis of such an analysis, we selected 30 frequently executed commands, listed in Table 1, and used them to construct job scripts, i.e., sequences of commands.

To obtain various levels, or intensities, of a host's load, we ran a variable number of jobs in the background. The artificial workloads were not intended to represent the typical workloads of personal workstations, but rather those of small (i.e., not very powerful) time-sharing systems. Workstations were used because of their being available in our distributed systems laboratory. We simulated user think times by the "*sleep*" command. The scripts are classified into three levels: light (L), moderate (M), and heavy (H), with a number of distinct scripts constructed for each level, so that hosts subjected to the same level of workload always use different scripts. The ranges of CPU utilization and mean load index values of the three levels of scripts are shown in Table 2. Each script runs for about 30 minutes on a Sun-2 workstation. Job and system performance statistics, such as

† For our experiments, to obtain accurate values of resource queue lengths and to perform the smoothing operations efficiently, some code had to be added to the kernel. No functional changes were made, however.

Table 1. Commands used in scripts and their eligibilities for remote execution

command	elig.	function	command	elig.	function
cat	N	*view a file*	*ls*	N	*directory listing*
cc	Y	*C compiler*	*man*	Y	*manual page viewing*
cp	N	*file copying*	*mv*	N	*move a file*
date	N	*current time*	*nroff*	Y	*text formatter*
df	N	*file system usage*	*ps*	N	*process checking*
ditroff	Y	*text formatter*	*pwd*	N	*current directory*
du	N	*disk usage*	*rm*	N	*delete a file*
egrep	Y	*text pattern search*	*sort*	N	*file sorting*
eqn	Y	*equation formatter*	*spell*	Y	*spelling checker*
fgrep	Y	*text pattern search*	*tbl*	Y	*table formatter*
finger	N	*user information*	*troff*	Y	*text formatter*
grep	Y	*text pattern search*	*uptime*	N	*system uptime*
grn	Y	*graph printing*	*users*	N	*list of current users*
lint	Y	*C program checker*	*wc*	N	*word count in a file*
lpq	N	*printer queue check*	*who*	N	*user information*

Table 2. Characterization of the workload levels

type	CPU utilization	average load index
light (L)	30-45%	0.3-0.7
moderate (M)	60-70%	1.0-1.8
heavy (H)	70-85%	1.8-3.0

resource demands, response times, resource utilizations, and resource queue lengths, were measured throughout each run. We used six Sun-2 workstations in our experiments.

As in any measurement experiment, we must consider the variability of the experimental environment, and therefore that of the measurement results. In dynamic load balancing, the placement of each job may vary from one run of the experiment to the next, because of the unavoidable variations in the timings of the events. This problem was further complicated in our experiments by the fact that we had to share the file server and the network with other parts of the research community. We tried to minimize this impact by running the experiments during the night. We repeated each experiment a number of times (typically 6), and computed the mean and the 90% confidence

interval (CI) of the values of the performance indices over these replications.

4. DESIGN AND RESULTS OF THE EXPERIMENTS

Experimental Factors

Four factors were identified to be of interest in the study of load indices:

1) **Load index.** We used as load indices the following quantities: the instantaneous CPU queue length; exponentially averaged CPU queue length; the sum of averaged CPU, file and paging/swapping I/O, and memory queue lengths†; and the average CPU utilization over a recent period. Inside the kernel, we kept variables for the queue length of each resource type. The length of each queue was sampled every 10 ms by the clock interrupt routine, and used to compute the one-second average queue length, q_i. Exponential smoothing was used to compute the average queue length over the last T seconds:

$$Q_i = Q_{i-1}(1-e^{-T}) + q_i e^{-T}, \qquad i \geq 1$$

$$Q_0 = 0$$

2) **Averaging interval** T. For exponentially smoothed values of a resource queue length, and for the average CPU utilization, the interval T over which the average is computed conceivably affects the quality of the index, and hence the system's performance.

3) **Workload.** There may be interactions between the load index chosen and the workload the system is subjected to. Using the three suites of host workloads described in the previous section, we were able to construct several combinations of system workload for the six workstations in our system. The canonical workload consisted of two heavy, two moderate, and two light scripts (2H, 2M, 2L). We also studied the indices under a more balanced workload, with all six workstations driven by moderate scripts (6M).

4) **Exchange interval** P. The GLOBAL algorithm employs periodic updates of load information. If P is too short, the overhead may be too high, but, if P is too long, then job placements are based on stale information, and performance may deteriorate, and system instability may result.

Measurement Results

We shall first study the indices and the averaging interval T by fixing the workload at its canonical level, and the exchange interval at 10 seconds. We will then use the more balanced workload 6M to examine the interactions between load indices and workload. Finally, we will study the effect of load exchange interval P on performance.

---

† For simplicity, we treated the disk queues as a single aggregate queue for I/O operations. For the memory queue, we identified a number of places inside the kernel where processes queue up for various types of memory resources (e.g., buffer space, page table), and treated all these as a single memory queue.

Table 3 shows the performance under various load indices. The numbers following the response time values indicate their 90% confidence intervals.

Table 3. Measured performance with various indices
(Canonical workload, $P = 10$ s)

replication count: 6
total number of jobs per run: 501
total number of eligible jobs per run: 254 (50.7%)
total number of processes per run: 766 (1.53 processes/job)
average process execution time: 7.45 s
approximate average CPU utilization for NoLB case: 60%

Load Index	Resp. Time	Improv.	Std. Dev.	Improv.
NoLB (no load bal.)	53.3 ±0.83	---	90.1	---
inst. CPU ql	35.0 ±0.68	34.4%	46.7	46.7%
1 s avg CPU ql	33.8 ±0.65	36.6%	45.8	49.2%
4 s avg CPU ql	33.1 ±0.39	37.9%	42.3	48.7%
4 s CPU+I/O+Mem ql	32.2 ±0.45	39.6%	44.3	50.9%
20 s avg CPU ql	37.0 ±1.20	30.6%	51.8	42.6%
20s CPU+I/O+Mem ql	35.6 ±0.12	33.3%	49.1	45.6%
60 s avg CPU ql	39.7 ±1.69	25.5%	54.1	40.0%
60s CPU+I/O+Mem ql	40.0 ±0.56	25.0%	56.2	37.6%
60 s UNIX load average	37.2 ±0.85	30.2%	54.9	39.1%
10 s CPU utilization	38.5 ±2.10	27.8%	55.4	38.5%
60 s CPU utilization	42.9 ±1.36	19.5%	67.6	25.0%

The indices can be divided into two categories: those based on resource queue length and those based on resource utilization. For each category, we can do the averaging over intervals of varying lengths.

We see in Table 3 that all the indices provide performance improvement, that is, they all contain some amount of *current* load information. The amount of improvement, however, varies quite widely: from 20% to 40%. This means that the performance of load balancing is heavily dependent on the load index used, and hence studying load indices is important. Comparing the two categories, the indices based on resource queue lengths are able to perform substantially better. This is probably because, when a host is heavily loaded, its CPU utilization is likely to be close to 100%; thus, in that region, the exact load level cannot be reflected by the value of the utilization. In contrast, queue lengths can directly reflect the amount of contention for a resource under heavy load. As an example, both a resource with an average queue length of 3 and one with a queue length of 6 probably have utilizations close to 100%, while they are obviously very differently loaded.

Comparing the queue-length-based indices with each other, we notice that the exponentially smoothed indices can perform best, but, if the averaging period T is too long (e.g., $\geq$ 20 s), performance may even become worse. Earlier in this paper, we have pointed out that, by averaging the queue lengths, the adverse effect of the high-frequency "noise" in the load can be reduced. This is reflected by improved performance. However, since the system load is changing all the time, averaging over too long a period will emphasize too much the past loads, which have little correlation with the future ones. The optimum averaging interval is clearly dependent upon the dynamics of the workload: the faster the load changes, the shorter the interval should be. In a measurement study of production workloads on a VAX-11/780 running Berkeley UNIX 4.2BSD [Zhou87b], we found that the average *net change* in CPU queue length in 30 seconds was 2.31, when the average CPU queue length itself was 4.12. This suggests that T should be substantially shorter than 30 seconds.

The performance difference between the cases in which indices based on CPU queue alone are used, and those in which indices consider I/O and memory contention also, is not significant, suggesting that the CPU is the predominant resource in our hosts. We found that the I/O and memory queue lengths were generally much shorter than that of CPU; that is, the former are much less contended for. It should be pointed out that our systems support general computing in a research environment; with other types of workload, e.g., database-oriented one, the contention profile of the various resource types may be substantially different. However, to achieve near-optimal performance, we do not have to consider all the resources in the system, but rather only those with significant contention. We also studied more general forms of linear combinations of queue lengths by using coefficients other than unity, but no significant changes in performance were observed. This, again, is probably due to the dominating influence of the CPU queue.

The load average shown in Table 3 is an index provided by a UNIX command; it is the exponentially smoothed number of processes ready to run, or running, or waiting for some high-priority event (e.g., disk I/O completion). A number of load balancers constructed in the past in the UNIX environment have used the load average as their load index (e.g., [Bershad85]). This research shows that significant further improvement can be obtained by using indices that more accurately reflect the current queueing at the resources.

The performances produced by the indices under the more balanced workload 6M is shown in Table 4. Since the workload is now more balanced and moderate, the amount of improvement in response time is not as much as that under the canonical workload; however, the relative rankings of the indices are quite similar. This suggests that the above analyses of the qualities of the indices and the appropriate values for T remain valid under a more balanced, moderate workload. It is worth noting that, in this case, due to the smaller improvement, using a poor load index (e.g., load average or 60 s CPU utilization) may yield little or no performance improvement.

Finally, we study the influence of the load exchange period P. Figure 2 shows the mean job response time as a function of P, and with the other three factors fixed. The brackets around the data points show their 90% confidence intervals. When the exchange period P is very short, the load information used in job placements is generally up to date, but this positive influence is outweighed by high message overhead. Conversely, if P is too long, the information may get stale, the quality of job placements deteriorates, and

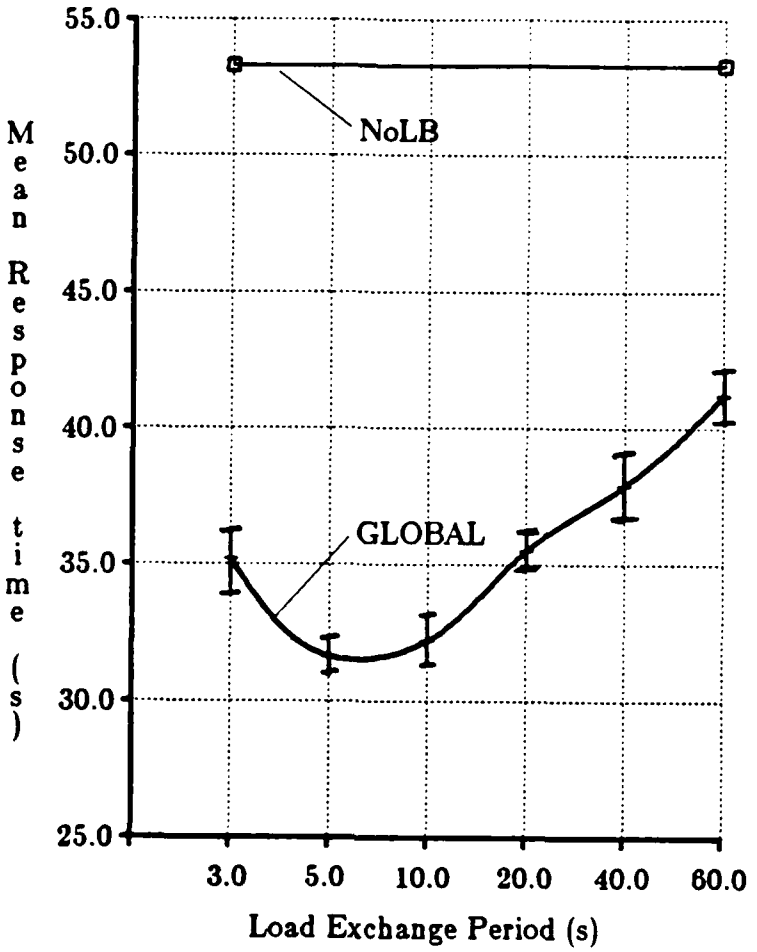

Figure 2. Mean process response time under various load exchange periods P (Canonical workload, load index 4 s CPU+I/O+Mem ql).

criteria reasonably well: the queue length is an accurate measure of a resource's load, and smoothing over a short interval into the past gives predictive capabilities to the value of the index, as well as stability against the noise in the load waveform. Queue-length-based load indices also appear to be more adaptable to a heterogeneous environment, but more studies are needed to substantiate this conjecture.

Our results support indices compatible with the one proposed in [Ferrari86], as they can be seen as degenerate forms of that index. However, the comparisons performed in this study are far from being complete. We decided to use the same load balancing algorithm for all the indices, so that the qualities of the load indices may be directly comparable. On the other hand, the algorithm limited the varieties of load indices that could be studied. We demonstrated, using a particular set of workloads and in a particular computing environment, that linear combinations of resource queue lengths may be good load indices. No proof, however, is offered that they are the best.

REFERENCES

[Alonso86]
R. Alonso, "Query Optimization in Distributed Databases through Load Balancing," Ph.D thesis, also as Tech Report, UCB/CSD 86/296, Computer Science Division, University of California, Berkeley, June 1986.

[Bershad85]
B. Bershad, "Load Balancing with Maitre d'," Tech Report, UCB/CSD 85/276, Computer Science Division, University of California, Berkeley, December 1985.

[Bryant81]
R. Bryant and R. Finkel, "A Stable Distributed Scheduling Algorithm," Proc. 2nd International Conf. on Distributed Computing Systems, pp. 314-323, 1981.

[Carey85]
M. Carey, M. Livny, and H. Lu, "Dynamic task allocation in a distributed database system," Proc. 5th International Conference on Distributed Computing Systems, Denver, May 1985.

[Chow79]
Y. Chow and W. Kohler, "Models of Dynamic Load Balancing in a Heterogeneous Multiple Processor System," IEEE Trans. Comp. Vol. C-28, No.5, pp. 354-361, May 1979.

[Eager86a]
D. Eager, E. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Dynamic Load Sharing," Performance Evaluation, Vol.6, No.1, pp. 53-68, April 1986.

[Eager86b]
D. Eager, E. Lazowska, and J. Zahorjan, "Dynamic Load Sharing in Homogeneous Distributed Systems," IEEE Trans. Soft. Eng., Vol.SE-12, No.5, pp. 662-675, May 1986.

[Ezzat86]
A. Ezzat, "Load Balancing in NEST: A Network of Workstations," Proc. 1986 Fall Joint Computer Conference, Dallas, TX, pp. 1138-1149, November 4-6.

[Ferrari86]
D. Ferrari and S. Zhou, "A Load Index for Dynamic Load Balancing," Proc. 1986 Fall Joint Computer Conference, Dallas, TX, pp. 684-690, November 4-6.

[Hac87]
A. Hac, "Load Balancing Algorithms for Distributed Systems," presentation at the Univ. of Calif., Berkeley, April 3, 1987.

[Hwang82]
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, Vol.15, No.4, pp. 55-66, April 1982.

[Joy80]
W. Joy, "An Introduction to the C Shell," Computer Science Division, University of California, Berkeley, November 1980.

[Joy83]
W. Joy, E. Cooper, R. Fabry, S. Leffler, K. McKusick, and D. Mosher, "4.2BSD System Manual," Computer Systems Research Group, University of California, Berkeley, July 1983.

[Lee86]
K. Lee and D. Towsley, "A Comparison of Decentralized Load Balancing Policies in Distributed Systems Characterized by Bursty Job Arrivals," Proc. 1986 SIGMETRICS Conference, pp. 70-77, May 1986.

[Leland86]
W. Leland and T. Ott, "Load-balancing Heuristics and Process Behavior," Proc. Performance '86 and ACM SIGMETRICS Conf on Measurement and Modeling of Computer

Systems, pp. 54-69, May 1986.

[Livny82]
M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," Proc. ACM Computer Network Performance Symposium, pp. 47-55, April 1982.

[MuKusick85]
K. McKusick, M. Karels, and S. Leffler, "Performance Improvements and Functional Enhancements in 4.3 BSD," Proc. Summer USENIX Conference, Jun 1985, Portland, OR, pp. 519-531.

[Wang85]
Y. Wang and R Morris, "Load Balancing in Distributed Systems," IEEE Trans. Comp. Vol.C-34, No.3, pp. 204-217, March 1985.

[Zhou86]
S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," Tech. Rept No. UCB/CSD 87/305, September 1986, also submitted for publication.

[Zhou87a]
S. Zhou and D. Ferrari, "An Experimental Study of Load Balancing Performance," Tech. Rept No. UCB/CSD 87/336 January 1987, also submitted for publication.

[Zhou87b]
S. Zhou, "An Experimental Assessment of Resource Queue Length as Load Indices," Proc. Winter USENIX Conference, Washington, D.C., pp. 73-82, January 21-24, 1987.

[Zhou87c]
S. Zhou, "Predicting Job Resource Demands: a Case Study in Berkeley UNIX," in preparation.