DYNAMIC LOAD BALANCING THROUGH PROCESS AND READ-SITE PLACEMENT IN A DISTRIBUTED SYSTEM

Anna Hać and Theodore J. Johnson

Anna Hać *is a member of technical staff in the Advanced Software Technology Department at AT&T Bell Laboratories (Indian Hill Park) in Naperville, Illinois, and* ***Theodore J. Johnson*** *is a graduate student at the Courant Institute of Mathematical Sciences at New York University. Ms. Hać works on performance evaluation and design of object-oriented systems and on distributed architectures for software switching applications. She has an M.S. and Ph.D. in computer science from the Technical University of Warsaw, Poland. Ms. Hać was a postdoctoral fellow at the University of California at Berkeley and, before joining AT&T in 1987, was an assistant professor of computer science at Johns Hopkins University. Mr. Johnson's research interests are concurrency control; parallel, distributed,* (continued on page 85)

This paper explains various methods for increasing performance in a distributed system. The focus of this study is load balancing and its relation to optimal process and read-site placement. The system model is based on the LOCUS distributed file system, which allows replicated files. The simulation system model includes process migration, and the CSS (centralized synchronization site) program enforces a synchronization policy. All requests to open a file for access must be sent to the file's CSS, which checks for access conflicts. An algorithm is provided that increases system performance through load balancing, basing its decisions on data collected by the system. We analyze and discuss the algorithm's characteristics and effects on system performance. Results show the load-balancing algorithm, if properly tuned, can improve performance.

Introduction

Most distributed systems are characterized by distribution of both physical and logical features. Distributed systems usually have a modular architecture. The system hardware, software, and data and the user software and data are distributed across the system. In addition, most distributed systems support a varying number of processing elements. An arbitrary number of system and user processes can be executed on various machines in the system.

When selecting a machine for process execution, such factors as resource availability and optimum use of resources [e.g., central processing unit (CPU), disk] must be considered. (Panel 1 identifies acronyms used in this paper.) In a distributed system environment, a load-balancing algorithm seeks the least busy machine. At the same time, the algorithm must not overload the system. Ideally, the algorithm uses available information to select the machine for process execution.

Panel 1. Acronyms in This Paper

ACM	Association for Computing Machinery
CSS	centralized synchronization site
CPU	central processing unit
FCFS	first-come, first-served
I/O	input/output
LAN	local-area network
SIGMETRICS	special interest group on metrics

Approaches to Load Balancing. The paragraphs that follow briefly describe several approaches to load balancing in a distributed environment. Each approach is further defined and described later in this paper.

In reference 1, Livny and Melman describe a *homogeneous distributed system* that has a high probability that at least one node is idle when tasks are queued at the other nodes. This example shows that the load-balancing algorithm is used less when the system is heavily loaded or when nodes are idle. In this paper, a model system with several nodes is described; each node has two servers with one queue. One communications channel connects the nodes. A simulation of this system shows that the load-balancing-algorithm process is a heavy user of the communications channel. Therefore, the amount of work done by the load-balancing algorithm may significantly decrease system performance.

In reference 2, Eager, Lazowska, and Zahorjan compare two strategies for *adaptive load sharing* with distributed control. The *sender-initiated strategy* allows congested nodes to search for lightly loaded nodes to which to move the processes. The *receiver-initiated strategy* allows lightly loaded nodes to search for congested nodes from which to transfer the processes. Ideally, sender-initiated strategies should be used in systems with light to moderate loads. Receiver-initiated strategies should be used in systems with high load, but only if the cost of process transfer is comparable under the two strategies.

In reference 3, Wang and Morris also compare several server-initiative and source-initiative algorithms and derive similar conclusions.

In reference 4, Tantawi and Towsley describe algorithms that determine the optimal load in a *heterogeneous distributed system* and construct a queueing network model of a system. In their system, a job may be processed at the host or transferred to another host. A job transfer causes communications delay and a queueing delay at the host that processes the job. However, the decision to transfer a job does not depend on the system state. Thus, the algorithm allows static load balancing.

In reference 5, de Souza e Silva and Gerla present an algorithm that finds the *optimum assignment of jobs* to sites in a distributed system. They show this static-load-balancing policy on an example of a queueing network model[6] of the LOCUS distributed system.[7]

In reference 8, Hać shows that file placement and process assignment improve system performance. The algorithm for a system with *distributed concurrency control* allows a file or process to be moved to the least loaded host. This decision is based on the number of read and write accesses to files, amount of use of the network server, and use of host resources.

Dynamic Load Balancing. This paper introduces a dynamic-load-balancing policy in a distributed system that consists of several hosts connected by a local-area network (LAN). This study's file system is modeled on the LOCUS distributed system.[9] The file system allows replicated files—that is, copies of each file may exist on several hosts—and provides a synchronization policy to update remote copies.

The simulation model allows process migration to different sites depending on a host's load. The algorithm implemented uses information collected in the system to choose a site for process execution and for a file-read access.

A token periodically entered into the system collects information about resource usage (CPU, disk) that is then used in the load-balancing strategy. The algorithm bases its decisions on various workload and system parameters. Because collecting the information is time-consuming and can overload the system, the algorithm also allows for the use of out-of-date information. The algorithm for dynamic load balancing tries to maximize performance in a distributed system by selecting the site for process execution and deciding on the read-site placement.

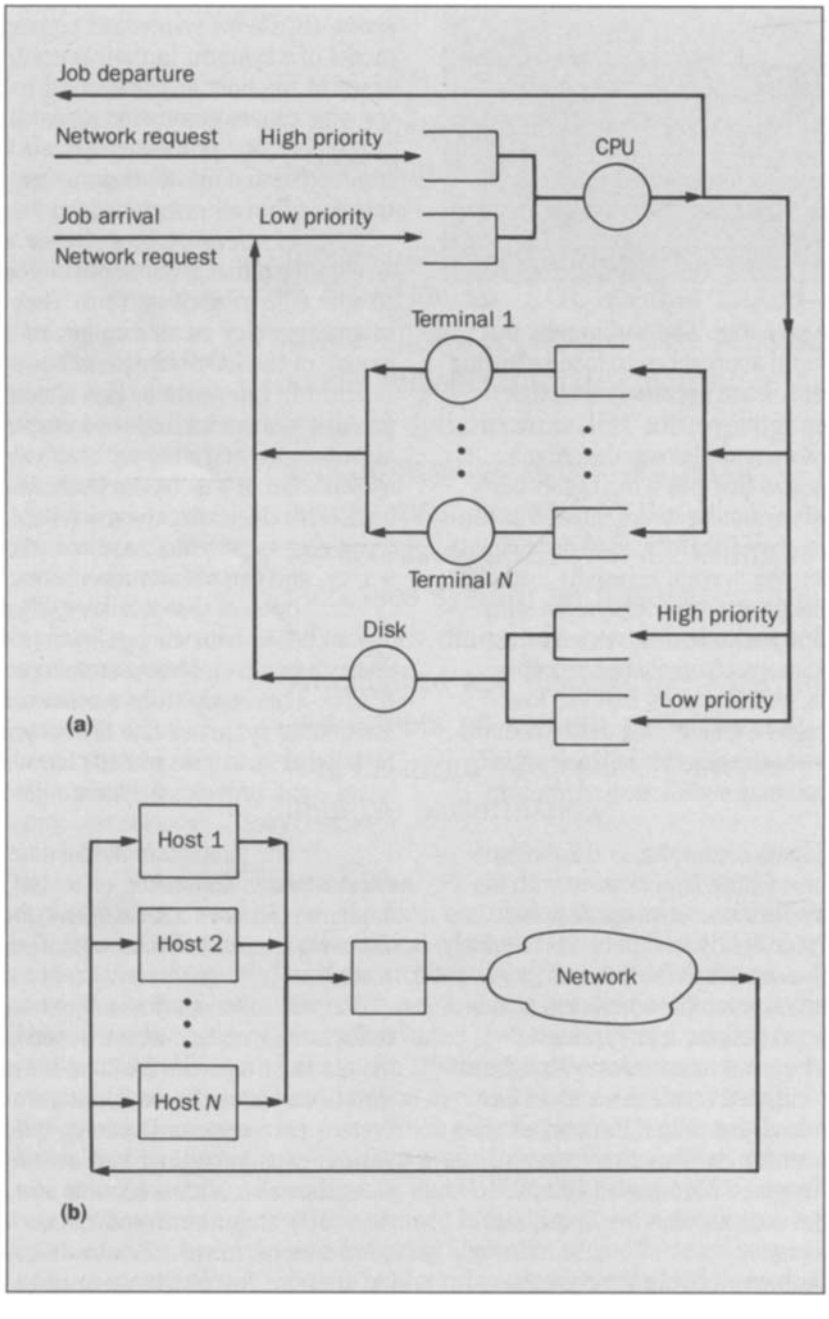

Figure 1. Model of a host (a) and of a distributed system (b) with *N* hosts. CPU means central processing unit.

Model of a Distributed System

As Figure 1 shows, the system is modeled by an open queueing LAN that consists of several interconnected queued servers. Each CPU has a round-robin service discipline; each is served in turn, in a fixed sequence. The disks and network have a FCFS (first-come, first-served) discipline; the first to request service is the first one served.

Service time distribution for the CPU and the disks is uniform. The network's service time distribution is deterministic but dependent on the size of the message transmitted. A job is entered into the system independently of the number of jobs already in the system. Jobs are submitted at a rate of three per second at each host. After visiting a sequence of servers, the job terminates. We implemented this model as an *event-driven simulator,* where an event in the system corresponds to an event in the simulation model.

The distributed file system is the most explored area of concentration in our study. Its synchronous policy is the one of multiple reader, single writer. To enforce this policy, we use the centralized synchronization site (CSS) program (as in the LOCUS operating system).[9]

This policy requires that every file have its own CSS. To open a file for access, a user request is sent to the file's CSS. If the request for access does not conflict with any current accesses, the request is granted. Otherwise, the request is refused. If the request to access has been granted, the requesting process may access the file. Once a process has finished its file access, the CSS must be notified so that it may update its file tables.

The file system supports replicated files, which increases a file's availability to users on other hosts if a host fails. Because a file may be replicated on several hosts, we had to implement a multiple-copy update mechanism so that all copies of a file that was updated could be brought into a consistent state.

For this task, we implemented a "pull" type mechanism; that is, each host with a copy of the file that a process altered is responsible for updating that copy. When a process writes to a file, it directs its updates to the file's primary site. After the file has been closed, all hosts that have a copy of this file are notified. They then activate update servers to read from the primary site and bring their own copies up to date.

To minimize the period during which copies of a file are inconsistent, the update servers run at high priority. We use dual-priority queues at the CPU and the disk to implement high priority for servers. A server with dual-priority queues searches the high-priority queue for a job before it searches the low-priority queue. The update servers use the high-priority queues while all other jobs use the low-priority queues.

Table I. Service Requirements

	Job service requirements (ms)		
Job type	CPU	Disk	Network
1	170.6	0.0	0.0
2	281.0	450.0	21.2
3	169.0	225.0	10.7
4	200.0	0.0	0.0
5	336.0	0.0	0.0
6	473.0	900.0	42.2
7	265.0	450.0	21.2
8	414.0	0.0	0.0

A Workload Model

Requests for job execution are scheduled for each host independent of the request schedule at other hosts. Once a job execution request has been sent to a host, the next execution request is scheduled to occur after a period determined by a sample from a uniform random variable.

We chose an arbitrary uniform distribution for interarrival times. To cover a wide range of possible jobs in a real system, we specified eight different job types, each with different service requirements. Table I lists the total amounts of service time at the CPU, disk, and network for each job type. The ratio of read-to-write disk accesses is 1:1.

The diagrams in Figure 2 illustrate the paths taken by each job type. System workload is specified by the probabilities that there are jobs of each job type. We

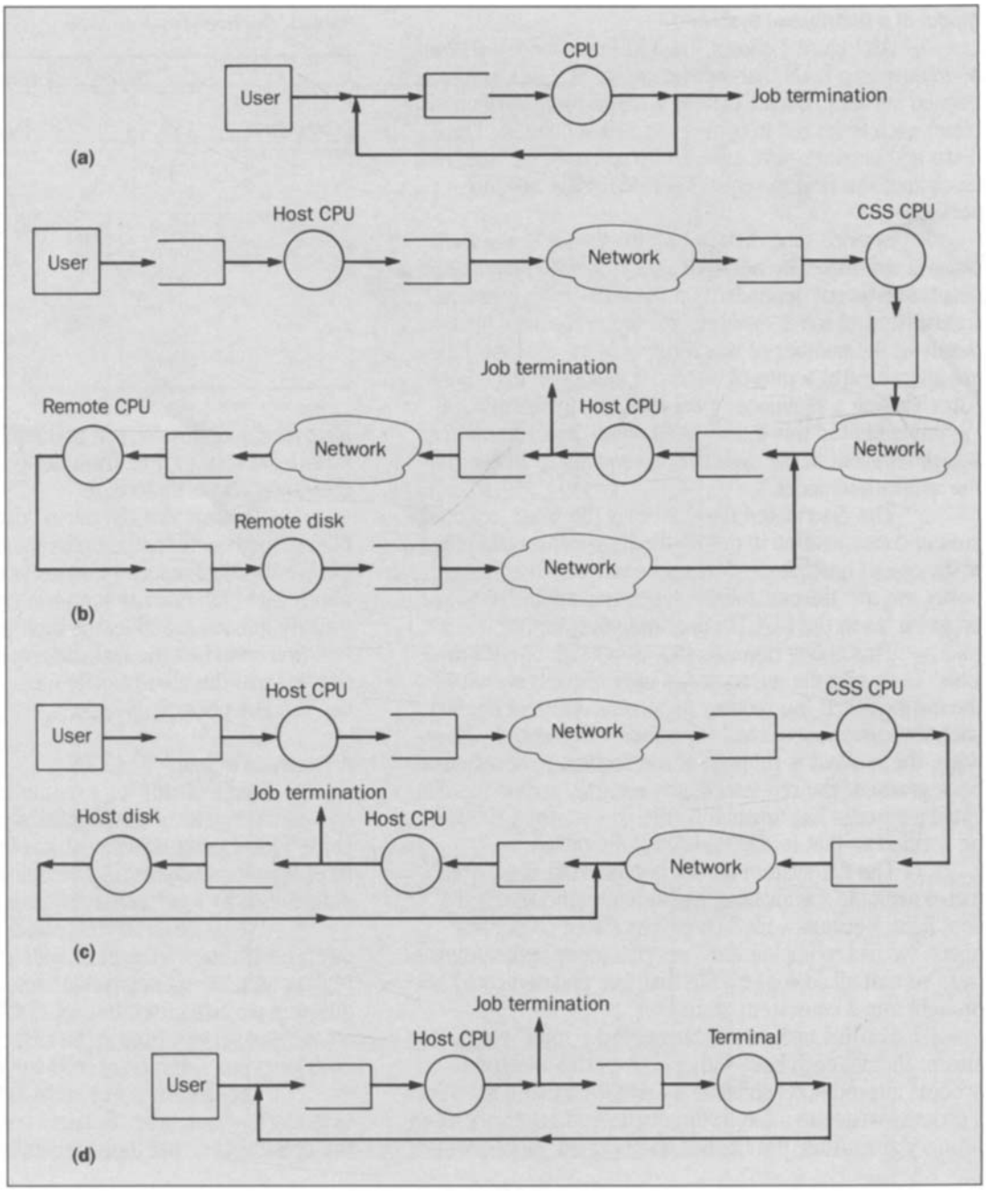

Figure 2. Graphs of the workload model for remote and local file accesses. CPU means central processing unit; CSS is the centralized synchronization site program. (a) Job types 1 and 5. (b) Remote file access for job types 2, 3, 6, and 7. (c) Local file access for job types 2, 3, 6, and 7. (d) Job types 4 and 8.

chose three workloads to cover the range of service requirements in a real system:

- *Workload C* describes CPU-intensive jobs and, therefore, a CPU-bound system in both nonsaturation and saturation operating modes. The system is in nonsaturation operating mode if an increase in the number of jobs of the same type also increases system throughput. Otherwise, the system is in saturation operating mode.
- *Workload D* describes I/O-intensive jobs and, therefore, an I/O-bound system in both nonsaturation and saturation operating modes.
- *Workload E* describes a mix of jobs that use CPU and disk servers equally.

Table II shows the average service requirements of each workload type.

The Simulator

In this study, we used an event-driven simulator written in the C programming language. In an event-driven simulator, events in the real system—request, allocate, and release a resource—are matched by events in the simulator. The system is simulated by scheduling the sequence of events for each job running in the system. To build the simulator, we determined the sequence of servers, or chain, that each job type will visit and specified the servers to accommodate the chains.

A simulation run was about 80 seconds long and completed 600 to 700 jobs. The confidence level of the simulations was 90 percent, and the width of the confidence interval was plus or minus 10 percent.

Implementation of Load Balancing

For load balancing, we implemented a read-site-placement mechanism at the CSS and a process-placement mechanism. While the LOCUS architecture manual[9] states that remote-process placement and process migration are implemented, we found no mention of a strategy for doing them to balance system loading.

Our algorithm for load balancing bases its decisions on information about the distribution of work in the system. A token is periodically entered into the system, collects workload measurements taken on every host, and distributes this information to each host. These measurements are: queue length of each server, percentage of use, and number of jobs using the resource. In addition, CPU and disk measurements are taken for each host.

However, dynamic collection and distribution of workload information in a distributed environment uses system resources. We took a *circulating token* approach because linear growth in the system's size (as measured by the number of hosts) causes linear growth in the amount of overhead for distributing the information. Therefore, there is no additional work on a per-host basis. The interval of the token-based update mechanism depends on the system workload, and is chosen so that the cost of update (as measured by CPU and network cost) is negligible. For the experiments we present, this interval is about 5 seconds.

Dynamic load balancing is implemented by choosing job execution sites and sites for file-read accesses. When a request for execution of a job arrives in the distributed system, the execution site is selected to balance the load on all hosts. If a remote-host site is selected as the execution site, a message is sent to the remote host telling it to start the job. Otherwise, the job is started on the local host.

When a job sends a request to the CSS to read a file, the CSS chooses a read site from among sites where the file is replicated. The CSS informs the job to direct its read requests to that site.

Algorithm for Load Balancing

The algorithm that chooses the execution site and read site uses vectors of workloads and host characteristics for each host. A vector is constructed for each possible selection site. The vectors are computed so that the longest one indicates the worst selection choice. Therefore, the host with the shortest vector is chosen. When an execution site is selected, the selection sites include all hosts. When a read site is selected, the selection sites include all hosts that have a copy of the file to be read. However, the algorithm does not check if the read site contains the most current version of the file.

The vector's dimensions correspond to factors that indicate the optimality of that site for selection. These dimensions are scaled with weights used to tune the algorithm. We did this to reflect the relative importance of the

Table II. Average Service Requirements and Average Utilization

Workload type	Average service requirement (ms)			Average utilization (%)		
	CPU	Disk	Network	CPU	Disk	Network
C	242.9	141.8	6.8	80	40	5
D	263.2	405.0	19.1	60	88	14
E	289.3	270.6	11.9	73	64	9

dimensions and allow for differences in the ranges of the measurements. For example, from our experience with the simulator, *percentage of use* ranges from 0 to 100 percent, while *queue length* rarely exceeds 10 for various types of workload.

When calculating the selection site, the load-balancing algorithm considers two types of dimensions:

- *Workload characterization*—These dimensions correspond to information collected by the circulating token: queue length, percentage of use, and number of jobs using the resource. To calculate vectors for process placement, the algorithm considers measurements made on the CPU. For read-site placement, the algorithm considers disk measurements. After considering these dimensions, the algorithm implements load balancing. As a placement site, it chooses the host with the lowest values of these measurements—thus, the least loaded host.
- *System-work minimization*—These characteristics tell whether a local or remote resource is being requested. They reduce the amount of work a job causes the system, both in reduced network usage and fewer requests to remote hosts. Moreover, these characteristics reduce the number of choices on which to base a load-balancing decision. An example of this reduction is the restriction: The set of choices for process placement is limited to sites that have a copy of the file. This restriction keeps the algorithm from overloading the resource. However, the algorithm bases its decisions on information that does not have to be current (e.g., other jobs may have entered that site's queue after the information was collected).

When making placement decisions, the algorithm uses information collected by the system. We had to find a balance between more frequent, thus more accurate, data collection and the amount of overhead that data collection causes. Therefore, we cannot avoid applying the algorithm to data that is not the most current. Lightly loaded hosts are chosen most often as execution and read sites. If we emphasize workload dimensions too much, by the time new data arrives for the algorithm to use, these hosts may be heavily loaded. An unbalanced load in the system results. To avoid this, we use work minimization characteristics, described later in this section.

Process Placement. For process placement, we considered CPU queue length, CPU utilization, and number of jobs active at a host as the workload characteristics. Our study did not consider disk characteristics, because they only affect file access.

The work minimization characteristics for process placement are:

1. Is the host being considered for job placement the host that requested the job?
2. Is the job accessing a file, and is the file stored at the host being considered?
3. Is the job interactive (that is, it accesses a terminal), and is the terminal at the host being considered?

Characteristics 2 and 3 are intended to give greater weight to a local host's resources. Characteristic 1 restricts the number of jobs placed at remote hosts. It indicates whether a host is local, and sets a level of load unbalance that must be attained before a job is placed at a remote host. If we set this level too low, hosts that had light loads when the system workload data was collected now are

Table III. Turnaround Time and Improvement Using Work Minimization

	Turnaround time (ms)					
	No work minimization		With work minimization		Improvement (%)	
Workload type	File access	CPU job	File access	CPU job	File access	CPU job
C	1578	8508	1465	9410	7.1	−9.5
D	2936	7669	2892	6702	1.4	12.6
E	1796	9408	1705	8699	5.0	7.5

flooded by job requests. If we set the level too high, load balancing will not occur. Finding an optimal weight for characteristic 1 is a large part of the task of tuning the algorithm.

Read-Site Placement. The workload characteristics for read-site placement are: disk queue length, disk utilization, and number of jobs accessing a file on the disk. Because a file access takes disk time, for simplicity we do not consider CPU characteristics.

The work minimization characteristic for read-site placement checks if the file is stored locally.

Let h define the host being considered, and let $w(h)$ be the length of the vector of load to be assigned to h. Then, the host selection algorithm is:

1) *for every host h being considered as a placement choice,*
  2) *for every workload characteristic being considered,*
    3) *then* $w(h) = w(h) + ((\text{weight for workload characteristic}) * (\text{workload characteristic}))\uparrow 2$
  4) *for every work minimization characteristic being considered,*
    5) *if the host being considered does not meet the work minimization condition,*
      6) *then* $w(h) = w(h) + (\text{weight for work minimization condition})\uparrow 2$
7) *choose the host, k, such that* $k = \{k: w(k) = \min(w(h))\}$.

Steps 2 and 3 consider workload characteristics. The length of the vector is increased with the square of the value of the workload measurement. Thus, hosts with lower workload dimensions are more likely to be chosen. Steps 4 through 6 consider work minimization characteristics. If a host does not meet the work minimization condition (for example, if the file is local), the length of the host's vector is increased. Thus, these hosts are less likely to be chosen.

Tuning the Algorithm. We selected the algorithm described in this paper because a sum of dimensions is a flexible approach that allows us to explore several factors important in load balancing. By setting the weight for a certain dimension to zero, we remove that factor's influence from the placement decision process. Likewise, an increase in the weight assigned to a dimension increases the corresponding factor's influence. In this algorithm, the higher workloads correspond to increasingly less likely choices for job placement. The values of the workload characteristics are squared, so that higher workload measurements result in longer dimensions, with a quadratic increase.

This algorithm chooses the host for process placement and the host for read-site placement, but the workload characteristics and work minimization characteristics are different for process placement and read-site placement. In the simulation model, we choose weights for each workload and work minimization characteristic to tune the algorithm.

The next section is devoted to studying tuning

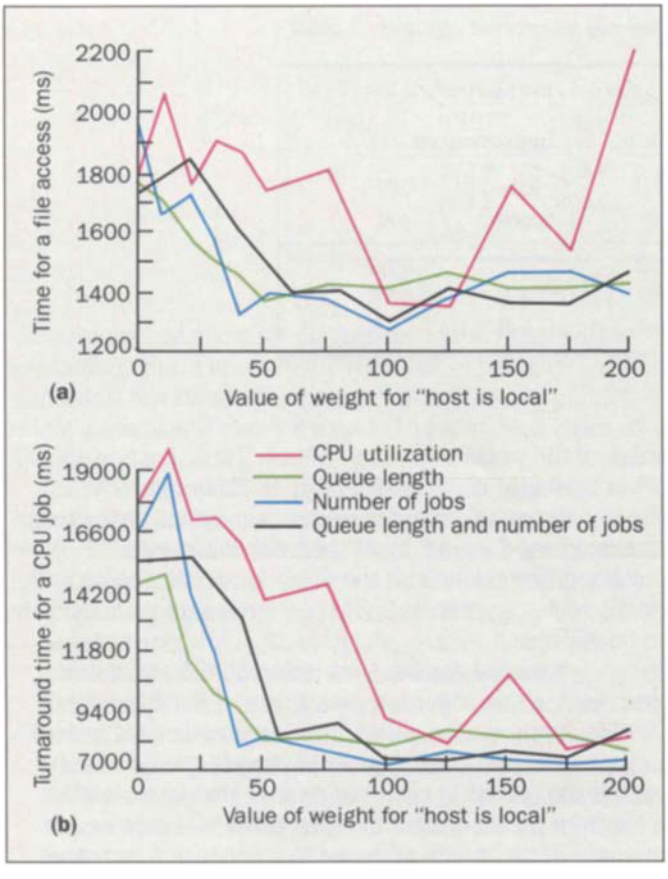

Figure 3. Access and turnaround times for the process-placement algorithm for workload C. (a) File access time; (b) turnaround time for a CPU-intensive job.

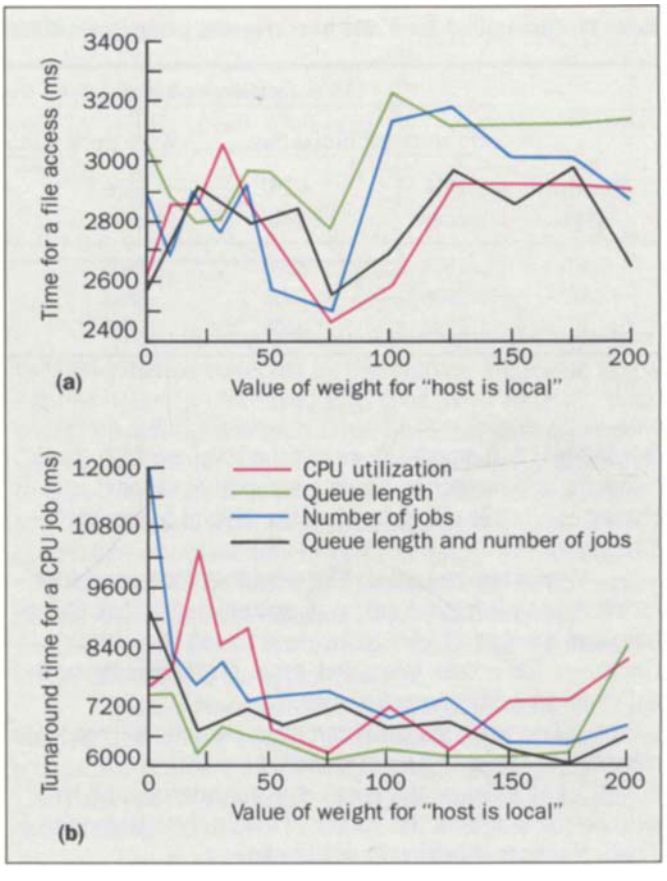

Figure 4. Access and turnaround times for the process-placement algorithm for workload D. (a) File access time; (b) turnaround time for a CPU-intensive job.

strategies. It concludes with a description of the vector of weights that yield the best turnaround times for each workload type studied.

Description of Experiments and Results

To determine the best placement sites, the load-balancing algorithm considers two types of characteristics when calculating vectors: work minimization characteristics and workload characteristics. If it uses only the work minimization characteristics, we can achieve performance improvement for all workload types, a result of job placement. Placement of job execution reduces work—remote resource (file, terminal) accesses and process migration—in the system. Table III presents a summary of improvements after work minimization characteristics were used in a system that consists of five hosts, where every file is replicated on two hosts.

Load balancing is implemented by using workload characteristics. They cause the algorithm to place jobs at lightly loaded hosts. This experiment determines a set of

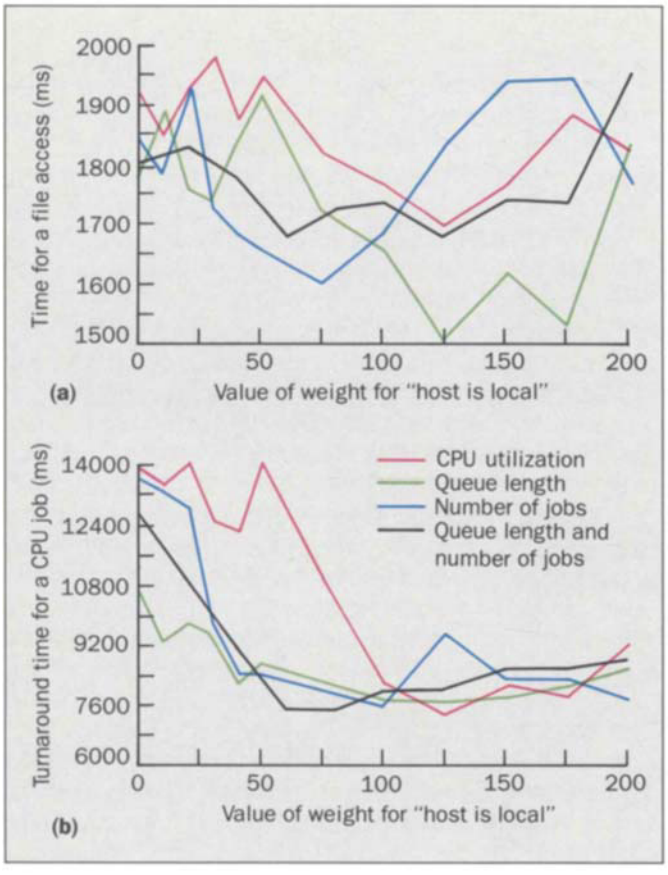

Figure 5. Access and turnaround times for the process-placement algorithm for workload E. (a) File access time; (b) turnaround time for a CPU-intensive job.

workload characteristics for the placement algorithm that causes performance improvement.

Two questions must be answered to determine the best method for implementing the load balancing:

- For a given type of workload, what is the best workload characteristic to use for load balancing? Because different workloads (for example, heavy or light, CPU-intensive or I/O-intensive) create different operating conditions, various workload characteristics may be better suited for certain uses.
- What is the best weights ratio for work minimization characteristics and workload characteristics? If we put too much emphasis on work minimization characteristics, the workload characteristics have less effect and load balancing is prevented from taking place. Thus, system performance on all hosts may not be optimal. However, if too much emphasis is put on workload characteristics, the system load may become unbalanced. For example, placing all processes on the least loaded host will unbalance the system. As a result, system performance is not optimized.

The Experiments. First, we conducted the following experiment. We configured a distributed system that consists of five hosts. In this system, every file is replicated on two hosts.

For study, we selected a particular workload characteristic (queue length, CPU or disk utilization, number of jobs, or queue length and number of jobs) for one of the placement types (process placement or read-site placement). This allowed us to compare the effectiveness of different workload characteristics for load balancing.

The weight for the selected dimensions was held constant, and all weights for other workload characteristics were set to zero, effectively removing their influence. We then varied one work minimization weight over a selected range to help determine the level at which to set that weight. The work minimization weight we selected for process placement was the characteristic: *host being considered for job placement is the host requesting the job* (i.e., work minimization characteristic 1, "host is local"). The work minimization weight for read-site placement was "file is local." We ran the simulator using weights for these characteristics for the placement algorithm, and executed the experiments for each type of workload.

Because the simulator uses an open-queueing-network model, jobs are added to the system regardless of how many jobs are already in the system. Turnaround times reflect system performance.

Results. For each simulator run, we recorded the time to make a file access and the time to complete a long CPU-intensive job. Table I displays the service requirements for this job type (job type 5). We chose these two turnaround times because they indicate performance for

both CPUs and disks. The time recorded for a file access is the time between the file open and close, divided by the number of file accesses. The time recorded for a long CPU-intensive job is the time between starting and ending the job.

Figures 3 through 5 show the results of the process-placement strategies for workload C (CPU-intensive), workload D (I/O-intensive), and workload E (mixed workload), respectively. Figures 6 through 8 shows the same results for the read-site-placement strategies. For each workload type, we plot file access time and turnaround time of a CPU-intensive job.

Process placement. One objective of this experiment was to determine the best balance between weights for the workload characteristics and weights for the work minimization characteristics. As Figures 3 through 5 show, turnaround times increase dramatically when the work minimization characteristic is set low. This places too much emphasis on the workload characteristics, which causes the algorithm to perform poorly. The characteristic that turnaround time is plotted against reduces the number of remote process placements. The difference between the workload measures for a remote host and a local host must be larger than the weight given to that characteristic before the job is sent to the remote host. If this weight is too low, jobs are placed at remote sites, instead of the more efficient local site.

On the other hand, giving too little emphasis to the workload characteristics prevents load balancing from taking place. Therefore, the difference in workload measures must be large enough that a remote job placement rarely occurs. In Figures 3 through 5, this occurs when the characteristic is set to a very high value. The turnaround times generally increase. Usually, turnaround times are an absolute minimum at some moderate value of the work minimization characteristic. For this value, balance between both workload characteristics and work minimization characteristics is the best.

Some of the plots tend to oscillate—for example, the I/O-intensive workload D (Figure 4). Load balancing for workload D is difficult to implement through process placement because of the large number of remote file accesses that occur in the system. But for workload D, we can achieve load balancing through read-site placement (Figure 7).

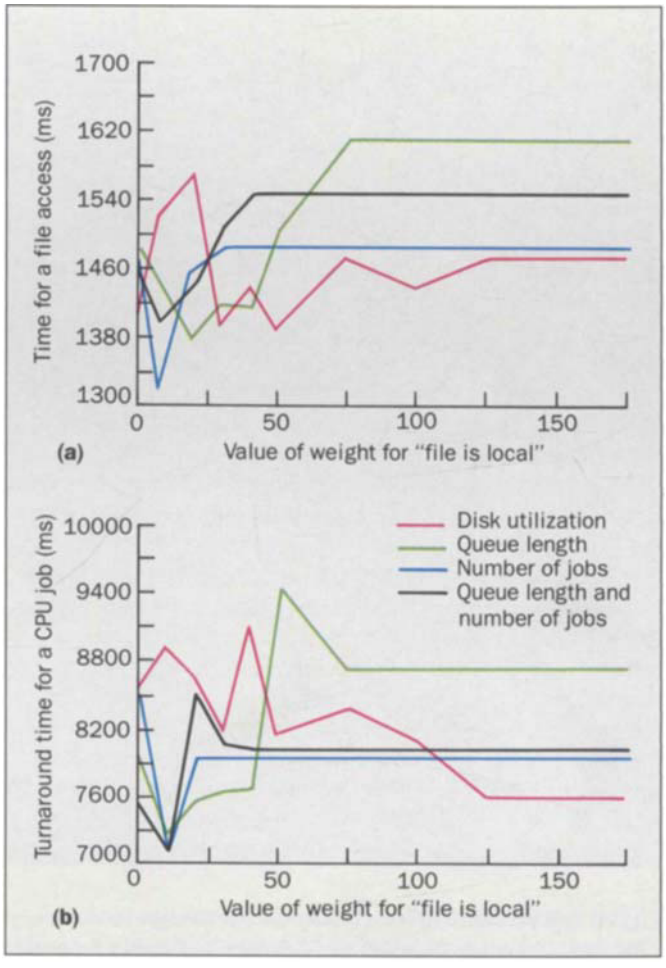

Figure 6. Access and turnaround times for the read-site algorithm for workload C. (a) File access time; (b) turnaround time for a CPU-intensive job.

In general, queue length and number of active jobs are nearly equally effective as workload characteristics. So, using both of them as workload characteristics is also effective. If we apply both characteristics, the algorithm becomes less sensitive to the work minimization character-

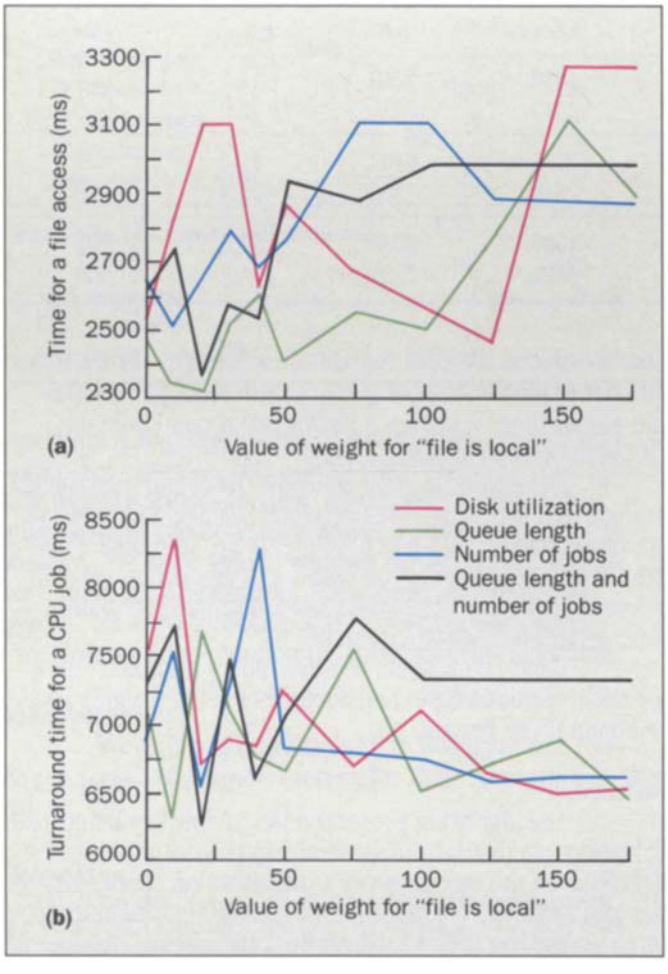

Figure 7. Access and turnaround times for the read-site algorithm for workload D. (a) File access time; (b) turnaround time for a CPU-intensive job.

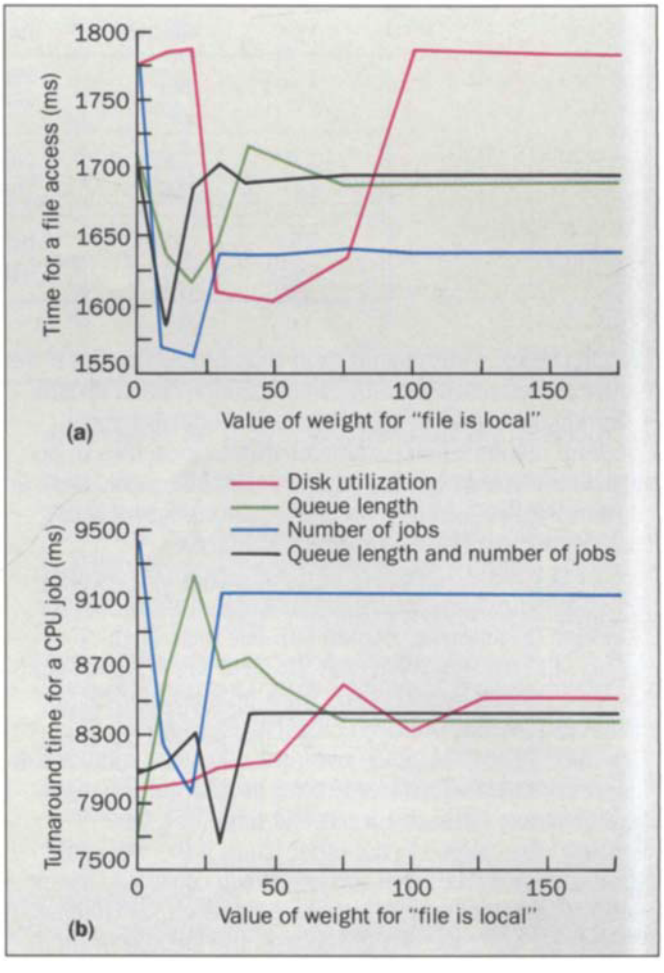

Figure 8. Access and turnaround times for the read-site algorithm for workload E. (a) File access time; (b) turnaround time for a CPU-intensive job.

istic. In Figure 5, the trough for the plot that corresponds to use of both characteristics runs between the troughs for plots that correspond to use of each characteristic alone.

Read-site placement. Figures 6 through 8 show the results of read-site placement strategies for workloads C, D, and E, respectively. Here, turnaround time of file access indicates system performance. Read-site placement directly affects file access time, but only indirectly affects the turnaround time of a CPU-intensive job. A trough in the graph usually appears when the value of the work minimization parameter is set low (for example, see Figures 6a, 7a, and 8a). This trough occurs because the number of read sites already restricts the number of hosts considered. We do not need to restrict the choices further

Table IV. Turnaround Time and Improvement Using Load Balancing

	Turnaround time (ms)					
	No load balancing		With load balancing		Improvement (%)	
Workload type	File access	CPU job	File access	CPU job	File access	CPU job
1 C	1578	8508	1236	7258	21.7	14.7
2 C	1578	8508	1410	7238	10.6	14.9
3 D	2936	7669	2476	6163	15.7	19.6
4 D	2936	7669	2544	7466	13.4	2.6
5 E	1796	9408	1589	8008	11.5	14.8
6 E	1796	9408	1671	7736	6.9	17.7

through use of work minimization characteristics. But if we remove the effect of the work minimization characteristic by setting its weight to zero, we degrade performance. The work minimization parameter causes local files to be chosen before remote files for read-site placement. If we ignore this consideration, we cause unnecessary remote file accesses, which degrades performance.

A noteworthy effect of the algorithm is that the plots level out when the work minimization characteristic is set high (Figures 6a, 7a, and 8a). For these high weights, the read sites become the local sites (Figure 6a). When we set the weight of the work minimization characteristic to 150, the workload characteristic affects placement if the read site is not local. File access times for high work minimization weights are usually not the most efficient times, because turnaround time for a CPU-intensive job is high. On the other hand, if the work minimization parameter is set too low, it can cause significant system performance degradation. As a result, at troughs in the plots for the time of a file access, the amount of time to complete a CPU-intensive job is usually high. (For example, look at the plot for the workload characteristic "number of jobs" in Figure 7 where the weight is set to 10.) Optimizing the turnaround time for one type of job at the expense of another is not usually acceptable. Here, the best choice for a work minimization weight is in the plot's flat area.

If disk usage is low, then disk utilization is the best choice as a workload dimension (e.g., for workload C, Figure 6, disk use is about 35 percent). For workload D, number of jobs and queue length are the best workload dimensions (Figure 7). For workload E, queue length and number of jobs are both reasonable selections. While using number of jobs instead of queue length results in a lower file access time, it causes a higher turnaround time for CPU-intensive jobs (Figure 8).

Tuning the placement algorithm. To obtain the results in Figures 3 through 8, we used load balancing for process placement or for read-site placement. So, to produce the best tuning strategy for the placement algorithm, we also executed experiments that use load balancing for both process and read-site placement.

Table IV presents the two most favorable results for each workload type, and Table V lists the weights used to obtain these results.

Conclusion

The algorithm presented here allows dynamic load balancing in a distributed system. This algorithm bases its decisions on the system's work minimization characteristics and workload characteristics. It uses vectors of loads on the hosts to choose the best host for process placement or read-site placement.

We executed several experiments to show applications of the algorithm for various workloads. These experiments show that different workload dimensions for the load balancing algorithm can be used to optimize system performance. Also, consideration of different process placement dimensions and read-site placement dimensions may improve system performance. The dimensions that promote the best performance for various workload types were chosen.

The experiments show that we can improve performance by using the load balancing algorithm. But if the

Table V. Weights for Workload Types in Table IV

Workload Dimensions	Workload type					
	1 C	2 C	3 D	4 D	5 E	6 E
Process placement dimensions:						
Host is local	100	100	75	25	125	60
Job is interactive	0	0	0	0	10	10
File is local	150	150	150	150	150	150
CPU queue length	0	0	0	13	13	13
CPU utilization	0	0	1	0	0	0
Number of jobs at CPU	7	7	0	0	0	7
Read-site placement dimensions:						
File is local	100	10	150	100	15	100
Disk queue length	0	0	0	20	0	20
Disk utilization	1	0	0	0	0	0
Number of jobs at disk	0	5	5	0	5	0

algorithm is improperly tuned, system performance may suffer. For example, this occurs when the work minimization weights are set too low. Fortunately, performance improvement can be achieved over a wide range of weights. We were able to improve the system's performance up to 21.7 percent for file access, and up to 19.6 percent for CPU-intensive jobs. These results provide incentive for implementing the load balancing algorithm.

Acknowledgment

We would like to thank Mike Gallagher and the *AT&T Technical Journal* referees for their comments on the manuscript.

References

1. M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," *Proceedings of the ACM Computer Network Performance Symposium,* College Park, Maryland, April 1982, pp. 47-55.
2. D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing," *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems,* Austin, Texas, August 26-29, 1985, pp. 1-3.
3. Y.-T. Wang and R. J. T. Morris, "Load Sharing in Distributed Systems," *IEEE Transactions on Computers,* Vol. C-34, No. 3, March 1985, pp. 204-217.
4. A. N. Tantawi and D. Towsley, "Optimal Static Load Balancing in Distributed Computer Systems," *Journal of the ACM,* Vol. 32, No. 2, April 1985, pp. 445-465.
5. E. de Souza e Silva and M. Gerla, "Load Balancing in Distributed Systems with Multiple Classes and Site Constraints," *Proceedings of Performance '84,* Paris, France, December 19-21, 1984, pp. 17-34.
6. A. Goldberg, G. Popek, and S. Lavenberg, "A Validated Distributed System Performance Model," *Proceedings of Performance '83,* College Park, Maryland, May 1983, pp. 251-268.
7. B. Walker et al., "The LOCUS Distributed Operating System," *Proceedings of the Ninth Symposium on Operating Systems Principles,* Bretton Woods, New Hampshire, October 10-13, 1983, pp. 49-70.
8. A. Hać, "File Placement and Process Assignment due to Resource Sharing in a Distributed System," *Proceedings of the Winter Simulation Conference,* San Francisco, California, December 11-13, 1985, pp. 481-492.
9. *The LOCUS Distributed System Architecture,* Edition 3.1, LOCUS Computing Corporation, 3330 Ocean Park Boulevard, Santa Monica, CA 90405, June 1984.

Biographies (continued)

and concurrent algorithms; and performance analysis. He has a B.A. in mathematics from Johns Hopkins University. The authors completed the work reported here and wrote the paper while at Johns Hopkins University.

(Manuscript received August 19, 1988)