A STUDY OF DYNAMIC LOAD BALANCING IN A DISTRIBUTED SYSTEM

Anna Hać and Theodore Johnson

JHU/EECS-85/15

A Study of Dynamic Load Balancing in a Distributed System

Anna Hać and Theodore Johnson

Department of Electrical Engineering and Computer Science
The Johns Hopkins University
Baltimore, Maryland 21218

ABSTRACT

This paper presents a study of a distributed system consisting of a number of identical hosts connected by a local area network. The system model is based on the LOCUS distributed file system. The LOCUS file system allows replicated files, and the synchronization policy is enforced by the use of the Centralized Synchronization Sites (CSS). All requests to open a file for access must be sent to the file's CSS which checks for access conflicts. Our simulation model allows process migration. The focus of this study is on load balancing as applied to optimal process and read site placement. An algorithm is proposed that increases system performance through load balancing. This algorithm uses data collected by the system to base its decisions on. The characteristics of the algorithm and their effects on system performance are analyzed and discussed.

1. Introduction

The distributed systems are characterized by distribution of both physical and logical features. The architecture of a distributed system is usually modular and consists of possibly varying number of processing elements. The system hardware and software, system data, user software and data, are distributed. An arbitrary number of system and user processes may be executed in the system. A process can usually be executed on various machines. There are a number of factors to be considered when selecting a machine for process execution. These factors may include resource availability and utilization of various resources. The least heavily loaded machine is usually chosen. An algorithm for load balancing should not overload the system, however. This is why an intelligent policy is needed that a machine for process execution can be selected based on an available information.

There are a number of approaches and solutions to load balancing in the distributed environment.

In [4] it is shown that in a homogeneous distributed system there is a high probability that at least one node is idle when tasks are queued at the other node. There is indicated that a load balancing algorithm should be used less when the system is heavily loaded, and also when the nodes are idle most of the time. A model is constructed for a number of nodes consisting of two servers with one queue, and one channel connecting the nodes. A simulation of this system shows that the balancing process is a heavy user of the communication channel. Therefore the amount of work done by load balancing algorithm has significant impact on the system performance.

Two strategies for adaptive load sharing with distributed control are compared in [1]. The sender-initiated strategy allows the congested nodes to search for lightly loaded nodes where the process may be moved. The receiver-initiated strategy allows the lightly loaded nodes to search for congested nodes where the processes may be transferred from. The result is that the sender-initiated strategies should be used in the systems with light to moderate loads, and that the receiver-initiated strategies should be used in the systems with high load only if the costs of process transfer under the two strategies are comparable.

The similar conclusions are derived in [8], where a number of server-initiative and source-initiative algorithms are compared.

The algorithms given in [6] determine the optimal load in a heterogeneous distributed system. A queueing network model of a system is constructed. A job may be processed at the host or may be transferred to the other host. A transfer of a job causes a communication delay, and the queueing delay at the host which the job is processed on. The decision to transfer a job does not depend on the system state. Thus the algorithm allows static load balancing.

The algorithm which finds the optimum assignment of the jobs to the sites in a distributed system is presented in [5]. This static load balancing policy is shown on an example of a queueing network model [2] of the LOCUS distributed system [7].

An improvement of system performance by file placement and process assignment is presented in [3]. The algorithm for a system with distributed concurrency control mechanism allows to move a file or a process to the least loaded host. The decision is made on the basis of the number of read and write accesses of the files, the utilization of the network server, and the utilization of the bottlenecks on all hosts.

In this paper we introduce a dynamic load balancing policy in a distributed system. The distributed system consists of a number of hosts connected by a local area network. The file system modeled in our study is the one of the LOCUS distributed system [9]. This file system allows replicated files, and provides a synchronization policy to update remote copies. Our simulation model allows process migration to different sites depending on the loads on the hosts. The implemented algorithm selects the site for process execution, and decides on the read site placement. This algorithm bases its decisions on the information collected in the system. A token is periodically entered into the system to collect the information of resource usage. This information is then used in our load balancing strategy. The algorithm bases its decisions on various workload and system parameters, and allows that information to be not necessarily up to date since the collection of the information is time consuming, and may also overload the system. The algorithm for dynamic load balancing attempts to maximize performance in a distributed system by selecting the site for process execution, and deciding of the read site placement.

2. The Model of a Distributed System

The system is modeled by an open queuing network as shown in Figure 1. An open queuing network is built of a number of interconnected queued servers. A job is entered into the system independently of the number of jobs already present in the system. After visiting a sequence of servers, the job terminates. This model is then implemented as an event driven simulator. In this simulator, 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. The synchronization policy is the one of multiple reader, single writer. This policy is enforced by the use of the Centralized Synchronization Sites (CSS) as in the LOCUS operating system, and defined in [9]. This policy requires that every file has a unique CSS associated with it. To open a file for access, a 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. After a process has finished the file access, the CSS must be notified so that it may update its file tables.

The file system supports replicated files (i.e., the files that may have copies existing on a number of hosts). The replication increases the file's availability to the users on other hosts in the case of a host failure. As a file may be replicated on a number of hosts, a multiple copy update mechanism has to be implemented so that all copies of a file may be brought into a consistent state after an update has been made. The multiple copy update mechanism implemented here is a 'pull' type mechanism. That is, the hosts with a copy of an updated file are responsible for bringing their own copies up to date. 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 with a copy of this file are notified. These hosts then activate update servers to read from the primary site and bring their own copies up to date. The update servers run at high priority, so as to keep the period during which copies of a file are inconsistent as short as possible. The high priority of the servers has been implemented by the dual priority queues at the CPU and at the disk. A server 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.

3. The Workload Model

The requests for job execution are scheduled for each host independently of the scheduling of requests at other hosts. After a job execution request is sent to a host, the next execution request is scheduled to occur after a period determined by a sample from a uniform random variable. Eight different job types are specified, each with different service requirements. The total amounts of service time at the CPU, disk, and network for each job type are given in Table 1. The graphs in Figure 2 illustrate the paths taken by each job type. The system workload is specified by the probabilities of jobs of each job type. Three workloads are chosen 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. 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 using CPU and disk servers equally. The average service requirements of each workload type are shown in Table 2.

Table 1. Service requirements for different types of jobs.

Job Type	Job Service Requirements [ms]		
	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

Table 2. Average service requirement and average utilization for different workload types.

Workload	Average Service Requirement [ms]			Average Utilization [%]		
Type	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

4. The simulator

The simulator used in our study is an event driven simulator written in the C programming language. In an event driven simulator, the events in the real system are matched by the events in the simulator. These events are: request, allocate, and release the resource. The system is thus simulated by scheduling the sequences of events for each job running in the system. The simulator is built by determining the sequence of servers, or chain, each job type will visit, and specifying the servers to accommodate the chains.

5. The Implementation of Load Balancing

The main area of concentration is on load balancing. The mechanisms for load balancing have been implemented in the Centralized Synchronization Sites (CSS), and as a process placement mechanism. While the LOCUS architecture manual [9] states that remote process placement and process migration are implemented, no strategy for doing so in regards to load balancing was mentioned.

Our algorithm for load balancing bases its decisions on the information about the distribution of the work in the system. A token is periodically entered into the system. This token collects and distributes to each host the workload measurements, taken by every host. These measurements are: the queue length, the utilization, and the number of jobs utilizing the resource. The resources in each host that these measurements are taken for are the CPU and the disk. The dynamic collection and distribution of information of the workload in a distributed environment utilizes system resources. The choice of the circulating token approach was thus made because a linear growth in the size of the system (as measured by the number of hosts) causes only a linear growth in the amount of overhead due to the distribution of the information. Therefore, there would be no additional work on a per host basis.

The dynamic load balancing is implemented by choosing job execution sites and by choosing sites for file read accesses.

When a request for a job execution arrives in the distributed system, the execution site is determined in order to balance the load on all hosts. If the execution site is chosen to be a remote site for a user on a host, a message is sent to the remote host informing it to start that job. Otherwise the job is started on the local host.

When a job sends a request to the CSS to read a file and the request can be granted, the CSS chooses a read site from the sites the file is replicated on. The CSS informs the job to direct its read requests to that site.

6. The Algorithm for Load Balancing

The algorithm which chooses the execution site and the read site uses vectors of loads on various hosts. A vector is constructed for each of the possible selection sites. When an execution site is selected, the selection sites include all hosts. When a read site is selected, the selection sites include all hosts with a copy of the file to be read from. The problem of whether the read site contains the current version of the file has not been addressed.

The dimensions of the vector correspond to the factors that indicate the optimality of that site for selection. These dimensions are scaled to reflect their relative weights, and to allow for differences in the scales in which measurements are taken. For example, utilization ranges from 0 to 100, while queue length rarely exceeds 10 for various types of workload as from experience with the simulator. The length of each vector is calculated, and the host whose vector has the shortest length is considered to be the best choice.

The algorithm considers two types of dimensions when calculating the site for selection.

First, there are the workload characterization dimensions. These correspond to the information collected by the circulating token: the queue length, the utilization, and the number of jobs utilizing the resource (i.e., CPU or disk). To calculate vectors for process placement, the algorithm considers measurements made on the CPU, while for read site placement, it considers disk measurements. It is through use of these dimensions that load balancing is implemented. As hosts with the lower values of measurements on these dimensions receive lower weights, they are more likely to be chosen. Hosts with larger values of measurements receive larger weights and are less likely to be chosen. Therefore, when given a choice, the algorithm will choose for a placement site the host with the lower values of measurements and thus the less loaded host.

Second, there are the system work minimization characteristics. These consider whether the resources being requested are local. These characteristics reduce the amount of work a job will cause the system both in reduced network usage, and by not requiring a remote host to respond to requests. Moreover, these characteristics intelligently reduce the number of choices to base a load balancing decision on. An example of this is to restrict the set of choices for process placement to the sites that have a copy of a file to be accessed. This keeps the algorithm from overloading the resource due to basing its decisions on the information which does not have to be the most up to date.

The algorithm uses information collected by the system when making placement decisions. A balance must be found between more frequent, thus more accurate, data collection, and the amount of overhead that collecting this data will cause. Therefore running the algorithm on the data which is not the most up to date cannot be avoided. As the most lightly loaded hosts will receive the lowest weights, they will be chosen most often. If too much emphasis is put on the workload dimensions, then by the time the new data arrives for the algorithm to use, these hosts will become heavily loaded. This will result in an unbalanced load in the system.

The workload characteristics considered for process placement are the CPU queue length, the CPU utilization, and the number of jobs active at a host. The disk characteristics are not considered because in our study they affect only the file access.

The work minimization characteristics for process placement are:

1) if the host being considered for job placement is the host requesting the job;

2) if the job accesses a file and the file is stored at the host being considered;

3) if the job is interactive (i.e., accesses a terminal), and the terminal is at the host being considered.

Characteristics 2 and 3 are intended to give greater weight to the hosts where resources are local. As the cost of remote job placement is relatively small, characteristic 1 is intended mostly to lessen the emphasis given to workload characteristics. This helps to keep the algorithm from overloading hosts due to basing its decisions on incomplete information.

The workload characteristics considered for read site placement are the disk queue length, the disk utilization, and the number of jobs accessing a file on the disk. As most of the time used in accessing a file is spent at the disk, CPU characteristics are not considered for simplicity.

The work minimization characteristic for read site placement is: if the file to be accessed is stored locally.

Let h define the host being considered. Let $w(h)$ be the length of the vector of load to be assigned to h. The algorithm to choose a host is:

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

Steps 2 to 3 consider workload characteristics. The length of the vector is increased proportionally to the value of the workload measurement. This makes hosts with lower workload dimensions more likely to be chosen. Steps 4 to 6 consider work minimization characteristics. A host not meeting the work minimization condition (the file is local, for example) has the length of its vector increased. This makes these hosts less likely to be chosen.

This algorithm chooses the host for process placement, and the read site placement. Note that the workload characteristic and the work minimization characteristic are different for process placement and read site placement. Therefore the different hosts may be chosen for process placement and read site placement using this algorithm. The algorithm is tuned by choosing the weights for each workload and work minimization characteristic. The next section is devoted to studying tuning strategies and concludes with the vector of weights that yields the best turnaround times for each workload being studied.

7. The Description of the Experiments and the Results

The load balancing algorithm considers two types of characteristics when calculating vectors to determine the best placement sites. These are the work minimization characteristics, and the workload characteristics. Using only the work minimization characteristics, the performance improvement may be achieved for all workload types. This is the result of job placement so that the job execution causes the minimum work in the system. Table 3 presents a summary of the improvements using only the work minimization characteristics. The load balancing is implemented by using the workload characteristics. These characteristics cause the algorithm to place jobs at lightly loaded hosts. The experiment presented here finds a set of workload characteristics for the placement algorithm that cause the performance improvement.

Table 3. Turnaround time and its improvement due to work minimization for different workload types.

Workload Type	Turnaround Time [ms] No Work Minimization		Turnaround Time [ms] With Work Minimization		Improvement [%]	
	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

In order to determine the best method for implementing the load balancing, two question have to be answered.

First, for a given type of workload what is the best workload characteristic to use for load balancing? As different workloads (i.e., heavy or light, CPU-intensive or I/O-intensive) create different operating conditions, various workload characteristics may be better to use.

Second, what is the best ratio between the weights for the work minimization characteristics and the workload characteristics? If too much emphasis is put on the work minimization characteristics, then this may prevent load balancing from taking place because of reducing the impact of the workload characteristics. Thus the system performance on all hosts may not be the best that can be achieved. However, if too much emphasis is put on the workload characteristics, then the algorithm may cause the system load to become unbalanced, for the reasons given in Section 6 (for instance, placing all the processes on the least loaded host). As the result the system performance may not be the best that can be achieved.

In order to answer these questions, the following experiment is conducted. A particular workload characteristic (the queue length, or the utilization, or the number of jobs, or the queue length and the number of jobs) for one of the placement types (process placement, or read site placement) is selected for study. The weight for the chosen dimension(s) is held constant and all weights for other workload characteristics are set to zero. One of the work minimization weights is varied over a selected range. The process placement work minimization weight selected is for the characteristic "host being considered for job placement is the host requesting the job" (work minimization characteristic 1). The read site work minimization weight is "file is local". The simulator is run using weights for these characteristics for the placement algorithm. The experiments are executed for each type of workload. The distributed system consists of 5 hosts. In this system every file is replicated on two hosts.

As the simulator uses an open queuing network model, jobs are added to the system independently of the number of jobs present in the system. The turnaround times are the indices of system performance. For each simulator run, the time to make a file access and the time to complete a long CPU-intensive job are recorded. These two turnaround times are chosen because they indicate the performance of both the CPUs and the disks. The time for a file access is the time between the file open and close, divided by the number of file accesses. The time for a long CPU-intensive job is the time between starting and ending the job.

The results for the process placement strategies are shown in Figures 3 to 8. Figures 3 and 4 show the results for workload C (CPU-intensive). Figures 5 and 6 show the results for workload D (I/O-intensive). Figures 7 and 8 show the results for workload E (mixed workload). For each workload, the time of file access and the turnaround time of a CPU-intensive job are plotted.

One objective of this experiment is to determine the best balance between the weights for the workload characteristics and the weights for the work minimization characteristics. In Figures 3 to 8, the turnaround times increase dramatically when the work minimization parameter is set to be low. Too much emphasis is put on the workload characteristics, causing the algorithm to perform poorly. The parameter that the turnaround time is plotted against has the effect of reducing the number of remote process placements. The difference between the workload measure of a remote host and the workload measure of the local host must be larger than the weight given to that parameter before the job is sent to the remote host. If this weight is too low, then jobs are placed at the remote sites while it would be better if they were at their local site.

On the other hand, giving too little emphasis to the workload characteristics prevents the load balancing from taking place. The difference in workload measures needs to be very large to cause a remote placement that it rarely happens. In Figures 3 to 8, this occurs when the parameter is set to a high value. The turnaround times generally increase. There is usually an absolute minimum at some moderate value of the work minimization parameter. At that minimum, the balance between the workload and work minimization characteristics is the best.

Note that some of the plots tend to oscillate. These are mostly the plots of workload D (I/O-intensive, Figures 5 and 6). Load balancing is difficult to implement in the system with workload D. This is due to the large number of remote file accesses occurring in the system. However, load balancing can be achieved here through read site placement (Figures 11 and 12).

In general, the queue length and the number of active jobs are nearly equally effective as workload characteristics. Using both the queue length and the number of active jobs as workload characteristics is also effective. Using both characteristics is better since it makes the algorithm less sensitive to what the value of the work minimization parameter needs to be. In Figures 7 and 8 the trough for the plot corresponding to the use of both characteristics runs between the troughs for the plots corresponding to the use of each characteristic alone.

The results obtained for read site placement strategies are shown in Figures 9 to 14. Figures 9 and 10 show the results for workload C (CPU-intensive). Figures 11 and 12 show the results for workload D (I/O-intensive). Figures 13 and 14 show the results for workload E (mixed jobs). The turnaround time of file access is the indicator of system performance. This is because read site placement directly affects file access time, but only indirectly affects a CPU-intensive job turnaround time. Note that a trough in the graph is usually when the value of the work minimization parameter is set to be low as in Figures 9, 11, and 13. The reason is that the number of read sites already restricts the number of hosts to be considered, and restricting the choices further through the use of work minimization characteristic is not necessary here. Removing the effect of the work minimization characteristic, however, (by setting its weight to 0) causes the performance degradation. The work minimization parameter causes local files to be chosen for read site placement before the remote files can be chosen. Ignoring this consideration causes significant work to the system in the form of unnecessary remote file accesses. The result is the performance degradation.

The characteristic of the algorithm to notice is that the plots level out after the weight for the work minimization characteristic is set high enough (Figures 9, 11, and 13). For these high weights the read site assignments become the local sites, if possible. An example is Figure 9, when the weight of the work minimization characteristic is set to 150. The workload characteristic has effect here if the read site is not local. File access times for high work minimization weights are usually not the best ones that can be obtained. The turnaround time for a CPU-intensive job is high. On the other hand, if the work minimization parameter is set too low, it may cause significant work to the system. As a result at the troughs in the plots for the time of a file access, the time to complete a CPU-intensive job is usually high. An example of this is the plot for the workload characteristic "number of jobs" in Figures 11 and 12 with the weight set to 10. Optimizing the turnaround time for one type of job at the expense of another is not usually acceptable. In this case, the best choice for a work minimization weight is in the flat area.

If the disk usage is low, then disk utilization is the best choice as a workload dimension (workload C, utilization is about 35%) as in Figures 9 and 10. For workload D, the number of jobs or the queue length are the best workload dimensions (Figures 11 and 12). For workload E, the queue length and the number of jobs are both reasonable selections. While using the number of jobs rather than the queue length results in the lower file access time, it causes a higher turnaround time of the CPU-intensive jobs (Figures 13 and 14).

The results shown in Figures 3 to 14 are obtained using the load balancing in process placement or read site placement. In order to produce the best tuning strategy for the placement algorithm, experiments were executed using the load balancing for both process placement and read site placement. The results of these experiments are shown in Figures 3 to 14. The two best results for each workload type are presented in Table 4. The weights used to obtain these results are presented in Table 5.

Table 4. Turnaround time and its improvement due to load balancing for different workload types.

Workload	Turnaround Time [ms] No Load Balancing		Turnaround Time [ms] With Load Balancing		Improvement [%]	
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

Table 5. Weights for process placement and read site placement dimensions for the workload types presented in Table 4.

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

8. Conclusion

The algorithm presented in this paper allows dynamic load balancing in a distributed system. This algorithm bases its decisions on the work minimization characteristic and the workload characteristic in the system. The algorithm uses vectors of loads on the hosts to choose the least loaded host for process placement or read site placement.

A number of experiments were executed to show the applications of the algorithm for various workloads. These experiments show that the different workload dimensions can be used to optimize the system performance using the load balancing algorithm. Also, the consideration of the different process placement dimensions and the read site placement dimensions, may improve the system performance. The dimensions which allow the best performance are chosen for various workloads.

The experiments show that the performance improvement can be achieved using the load balancing algorithm. However, if the algorithm is improperly tuned, the system performance may suffer, an example of this occurs when the work minimization weights are set too low. Fortunately, the performance improvement can be achieved with a wide range of weights. The performance of our system was improved by up to 21.7% for file access, and up to 19.6% for the CPU-intensive jobs. This provides incentive for implementing the load balancing algorithm.

9. References

[1] D. L. Eager, E. D. Lazowska and J. Zahorjan, A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing, Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (August 26-29, 1985, Austin, Texas) 1-3.

[2] A. Goldberg, G. Popek and S. Lavenberg, A Validated Distributed System Performance Model, Proc. Performance'83 (1983) 251-268.

[3] A. Hać, File Placement and Process Assignment due to Resource Sharing in a Distributed System, Proc. Winter Simulation Conference (Dec. 11-13, 1985, San Francisco, California).

[4] M. Livny and M. Melman, Load Balancing in Homogeneous Broadcast Distributed Systems, Proc. ACM Computer Network Performance Symposium (April 1982, College Park, Maryland) 47-55.

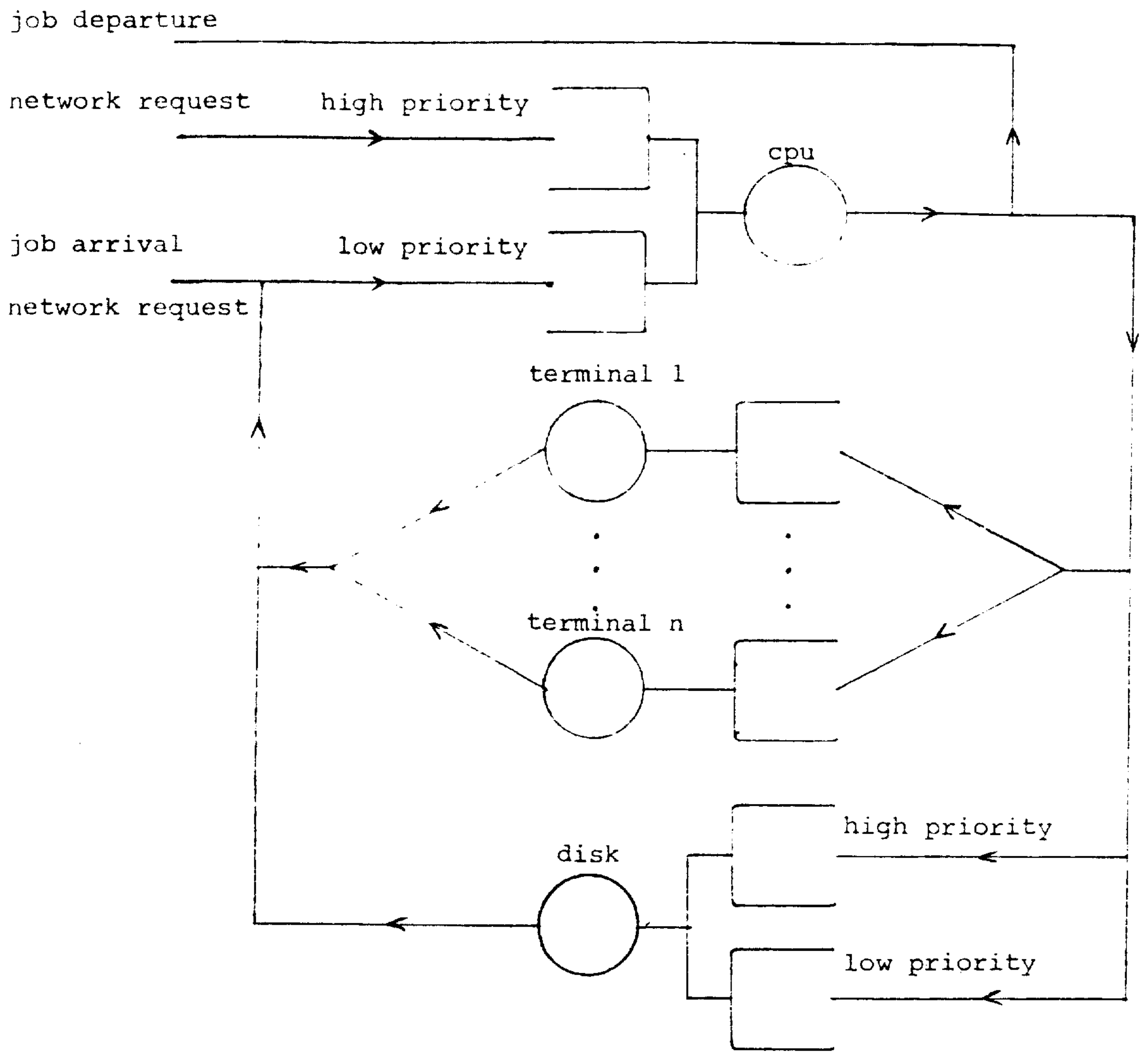
[5] E. de Souza e Silva and M. Gerla, Load Balancing in Distributed Systems with Multiple Classes and Site Constraints, Proc. Performance'84 (1984) 17-34.

[6] A. N. Tantawi and D. Towsley, Optimal Static Load Balancing in Distributed Computer Systems, Journal ACM, 32, 2 (1985) 445-465.

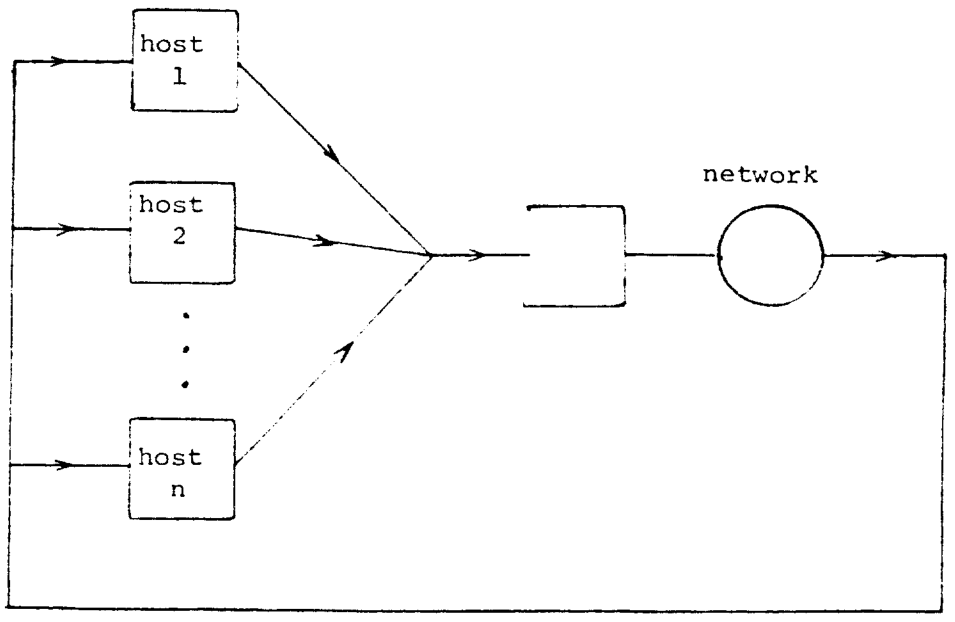
[7] B. Walker, G. Popek, R. English, C. Kline and G. Thiel, The LOCUS Distributed Operating System, Proc. Ninth Symposium on Operating Systems Principles (October 1983) 49-70.

[8] Y. -T. Wang and R. J. T. Morris, Load Sharing in Distributed Systems, IEEE Transactions on Computers, C-34, 3 (1985) 204-217

[9] The LOCUS Distributed System Architecture, Edition 3.1 (June 1984) LOCUS Computing Corporation, 3330 Ocean Park Blvd., Santa Monica, CA 90405.



a) the model of a host



b) the model of a distributed system

Figure 1. The model of a host and a distributed sytem.

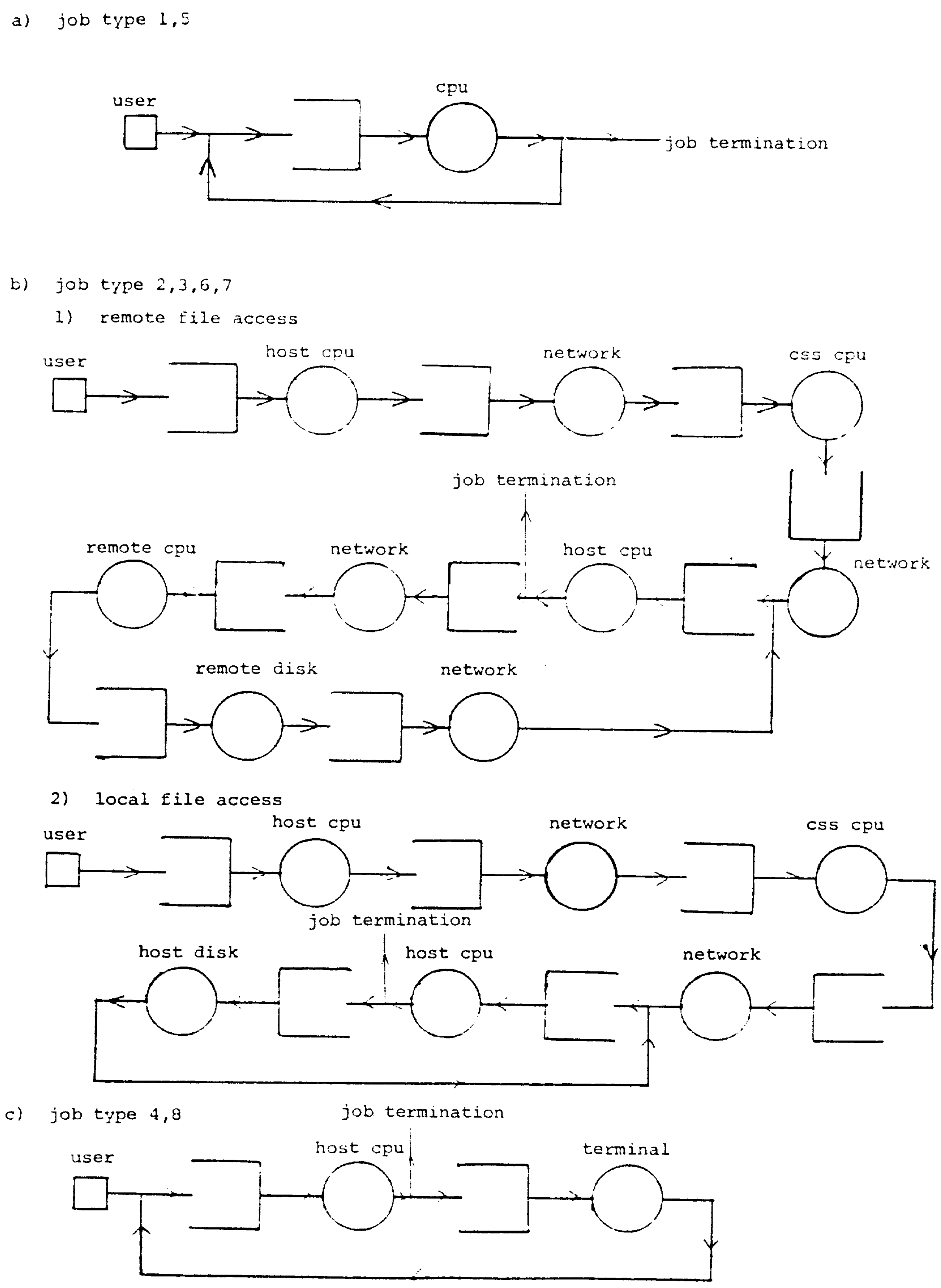


Figure 2. The graphs of the workload model.

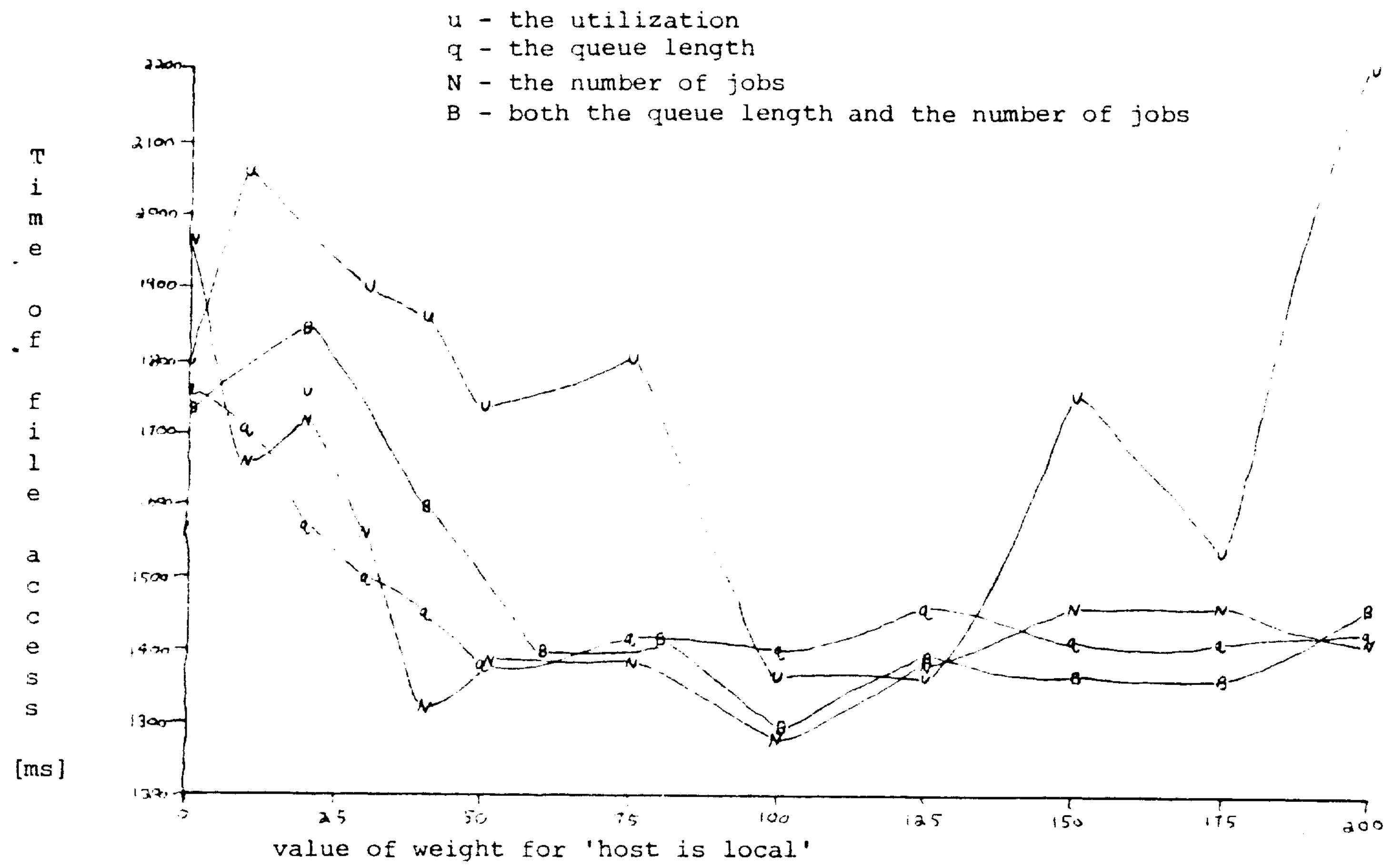


Figure 3. The time of a file access for the process placement algorithm for workload C.

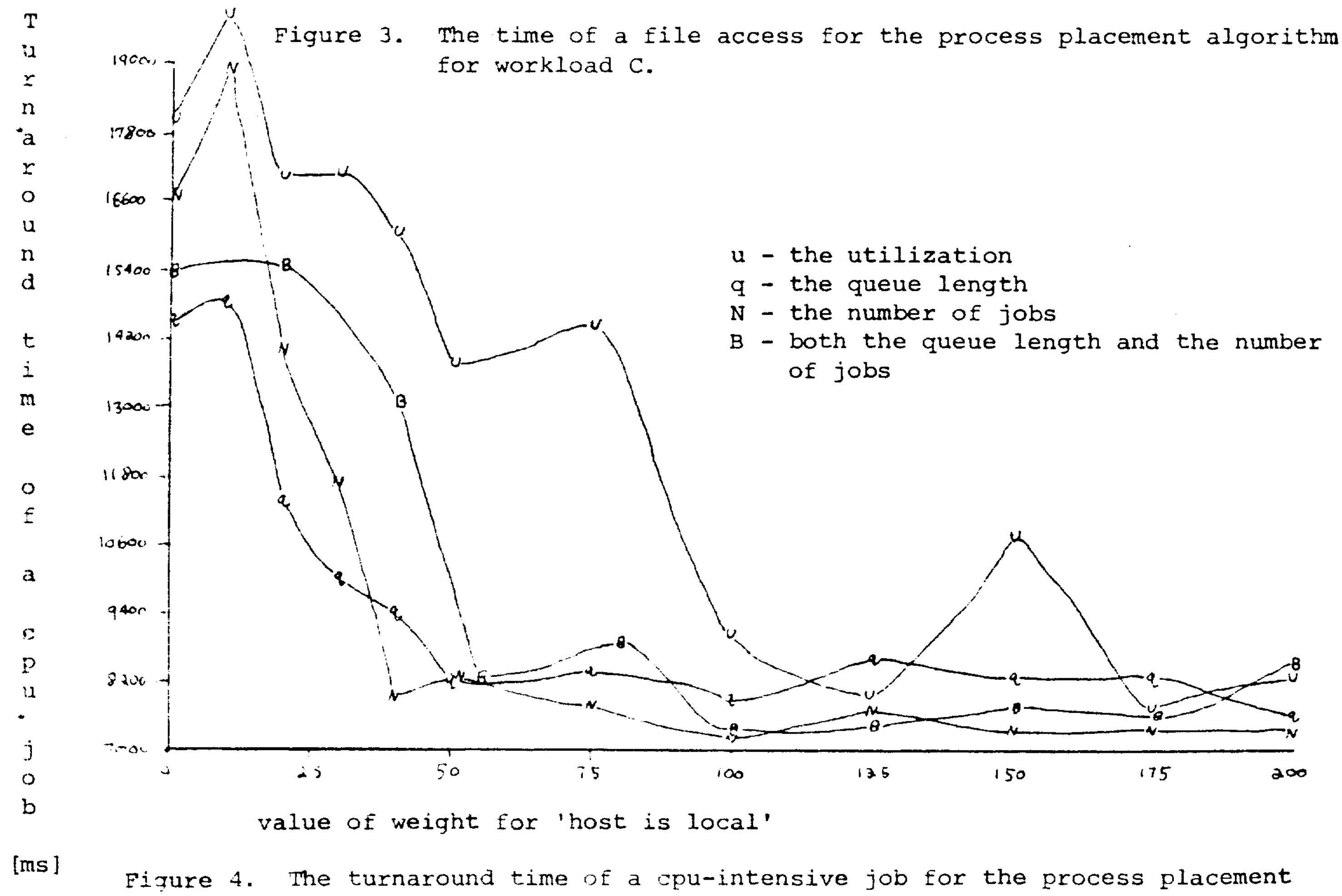


Figure 4. The turnaround time of a cpu-intensive job for the process placement alogrithm for workload C.

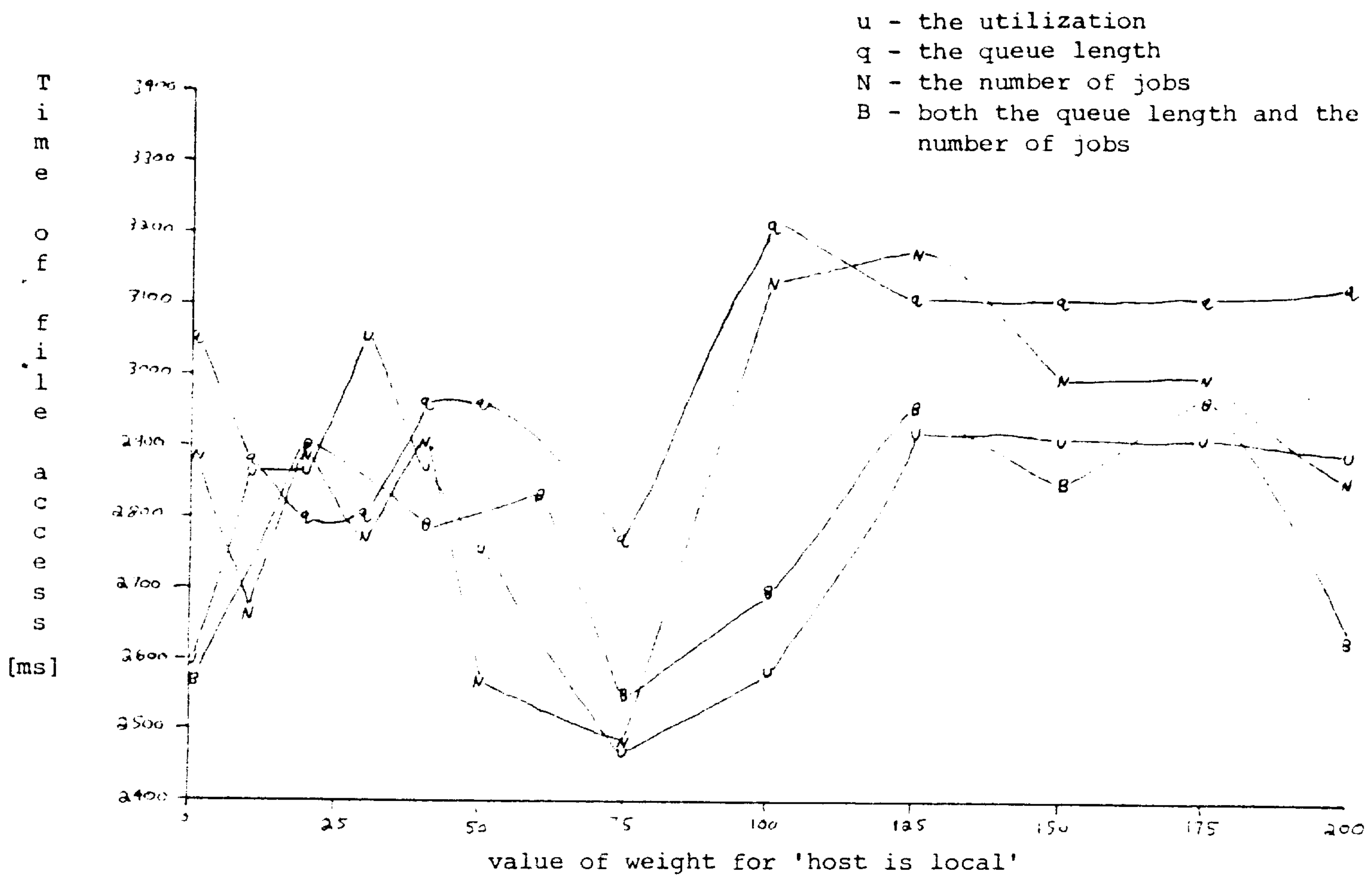


Figure 5. The time of a file access for the process placement algorithm for workload D.

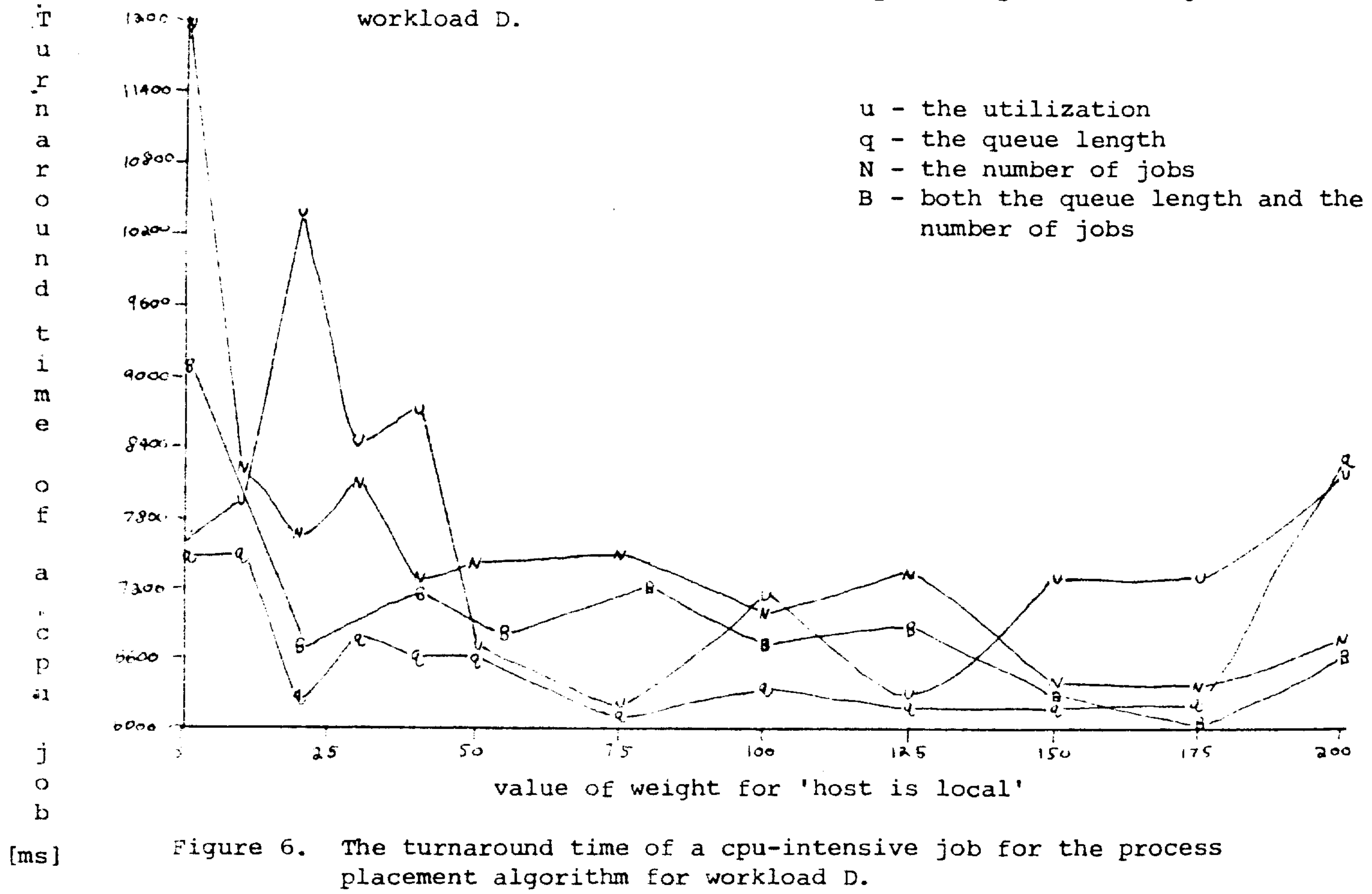


Figure 6. The turnaround time of a cpu-intensive job for the process placement algorithm for workload D.

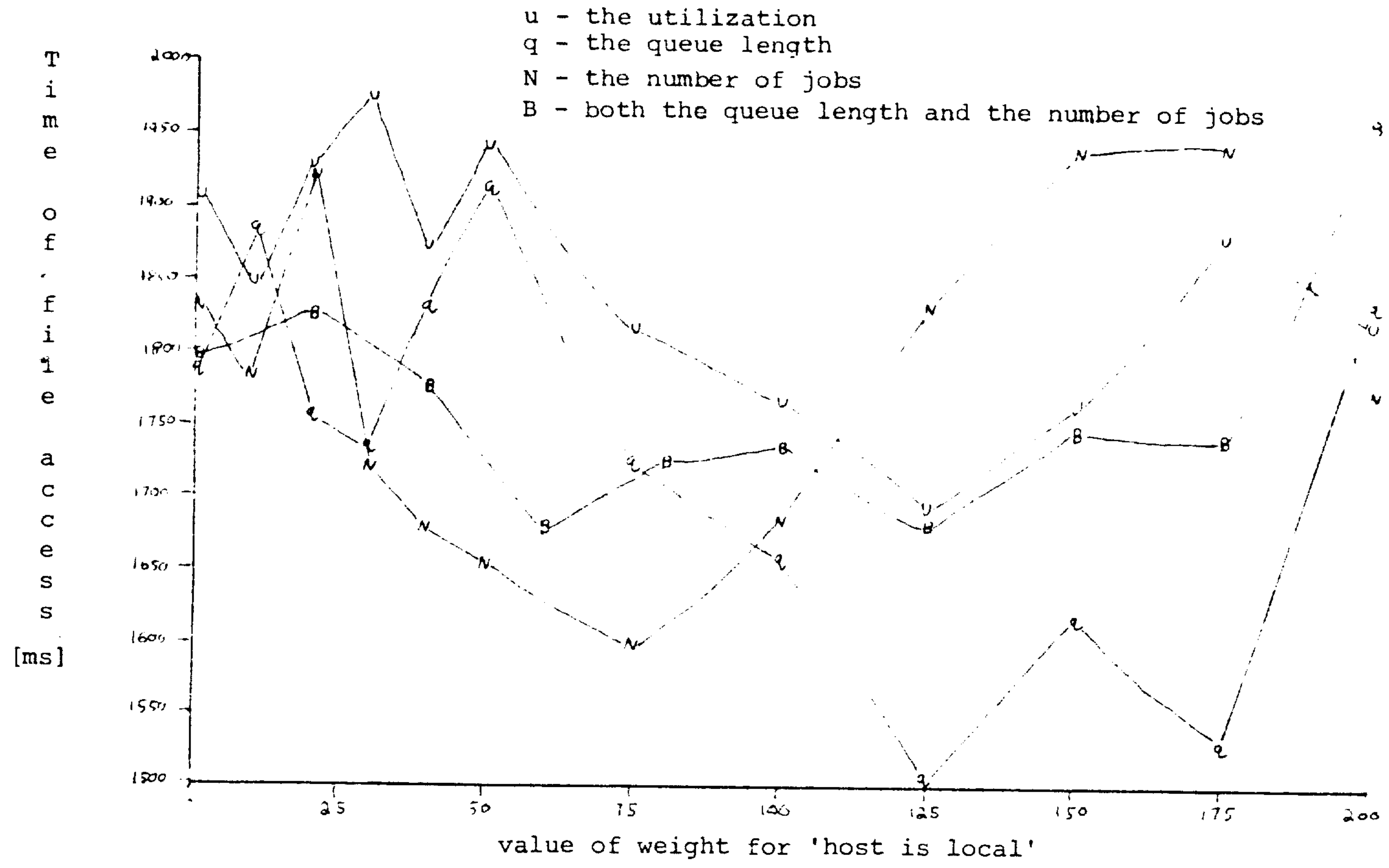


Figure 7. The time of a file access for the process placement algorithm for workload E.

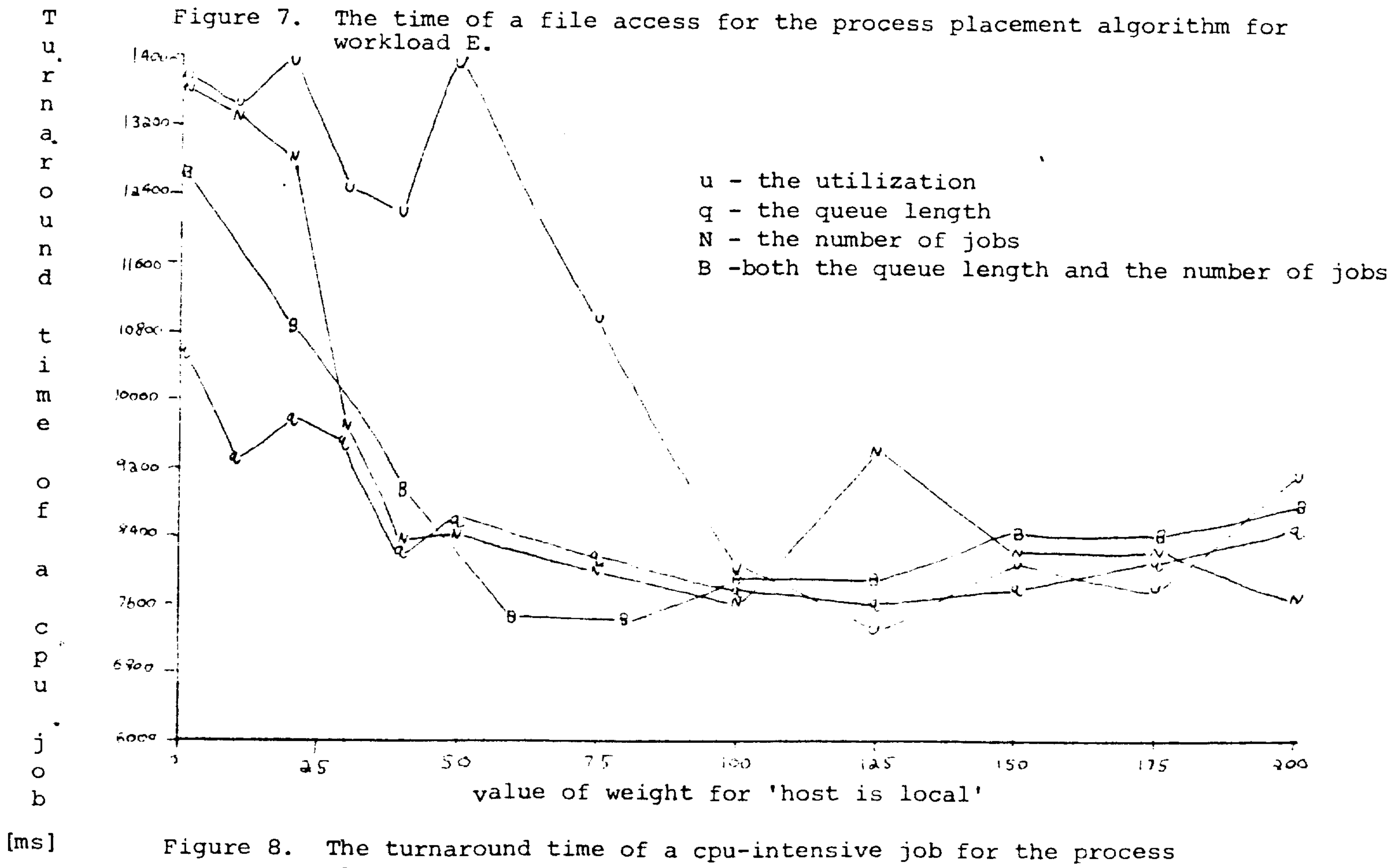


Figure 8. The turnaround time of a cpu-intensive job for the process placement algorithm for workload E.

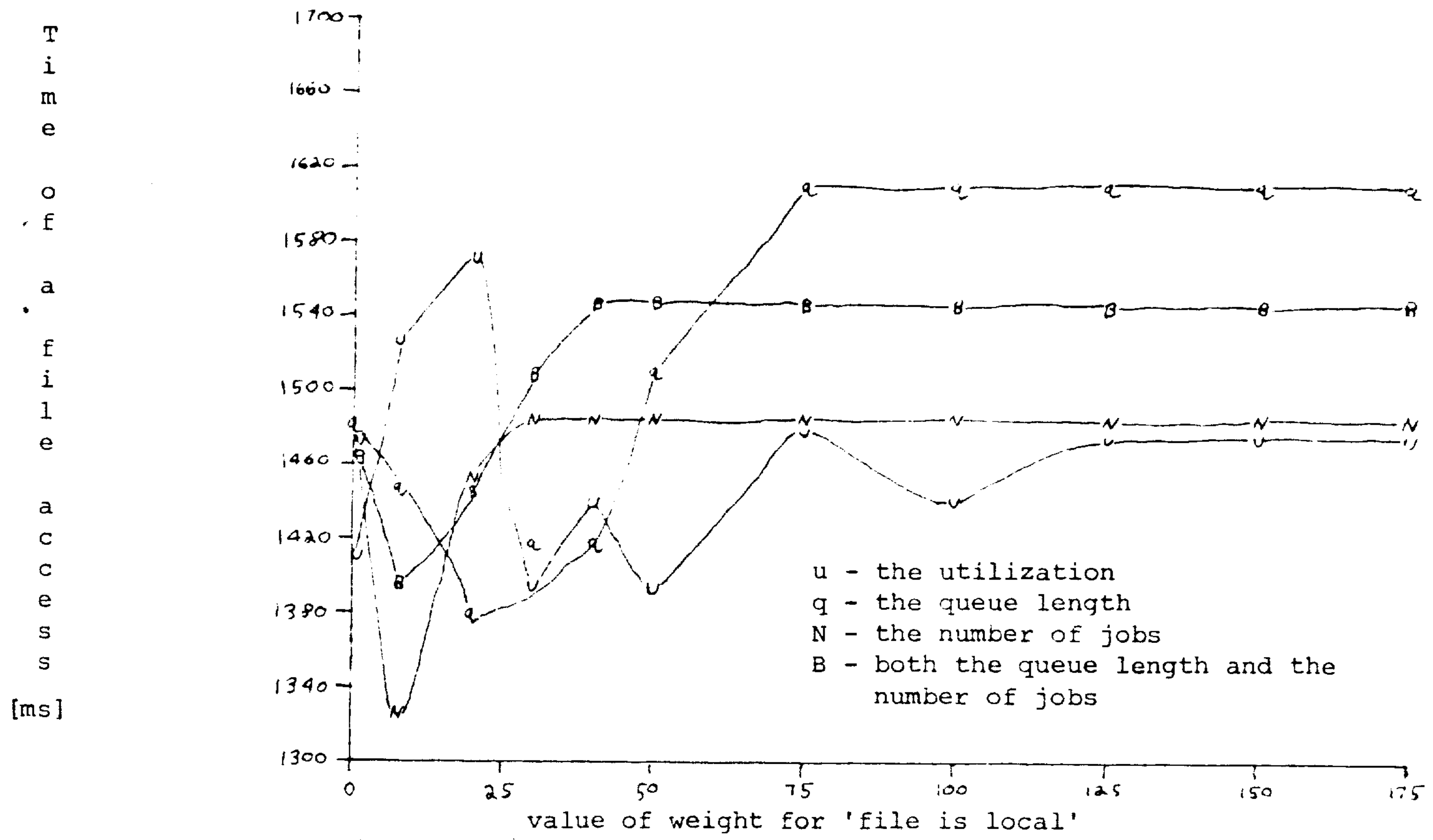


Figure 9. The time of a file access for the read site placement algorithm for workload C.

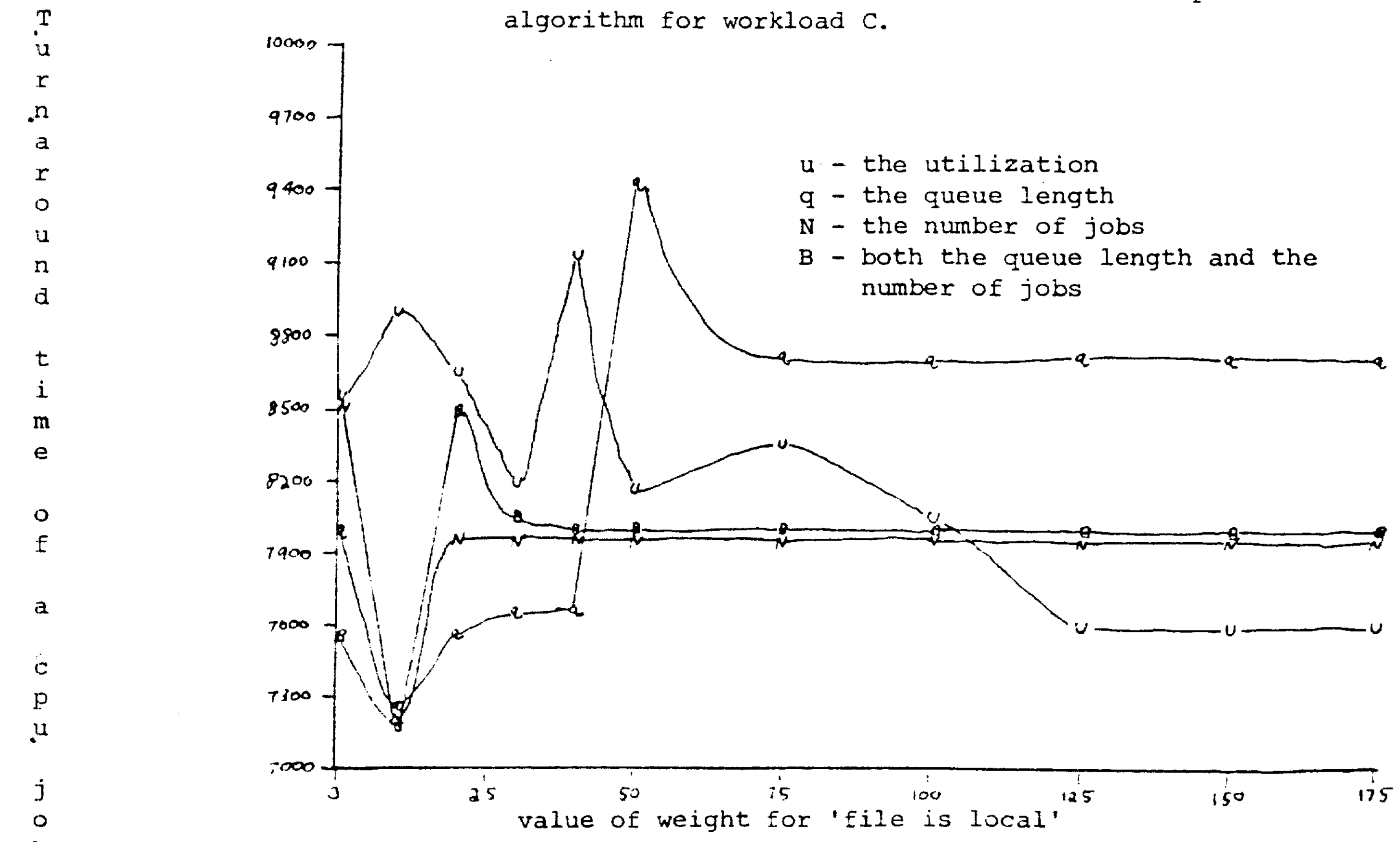


Figure 10. The turnaround time of a cpu-intensive job for read site placement algorithm for workload C.

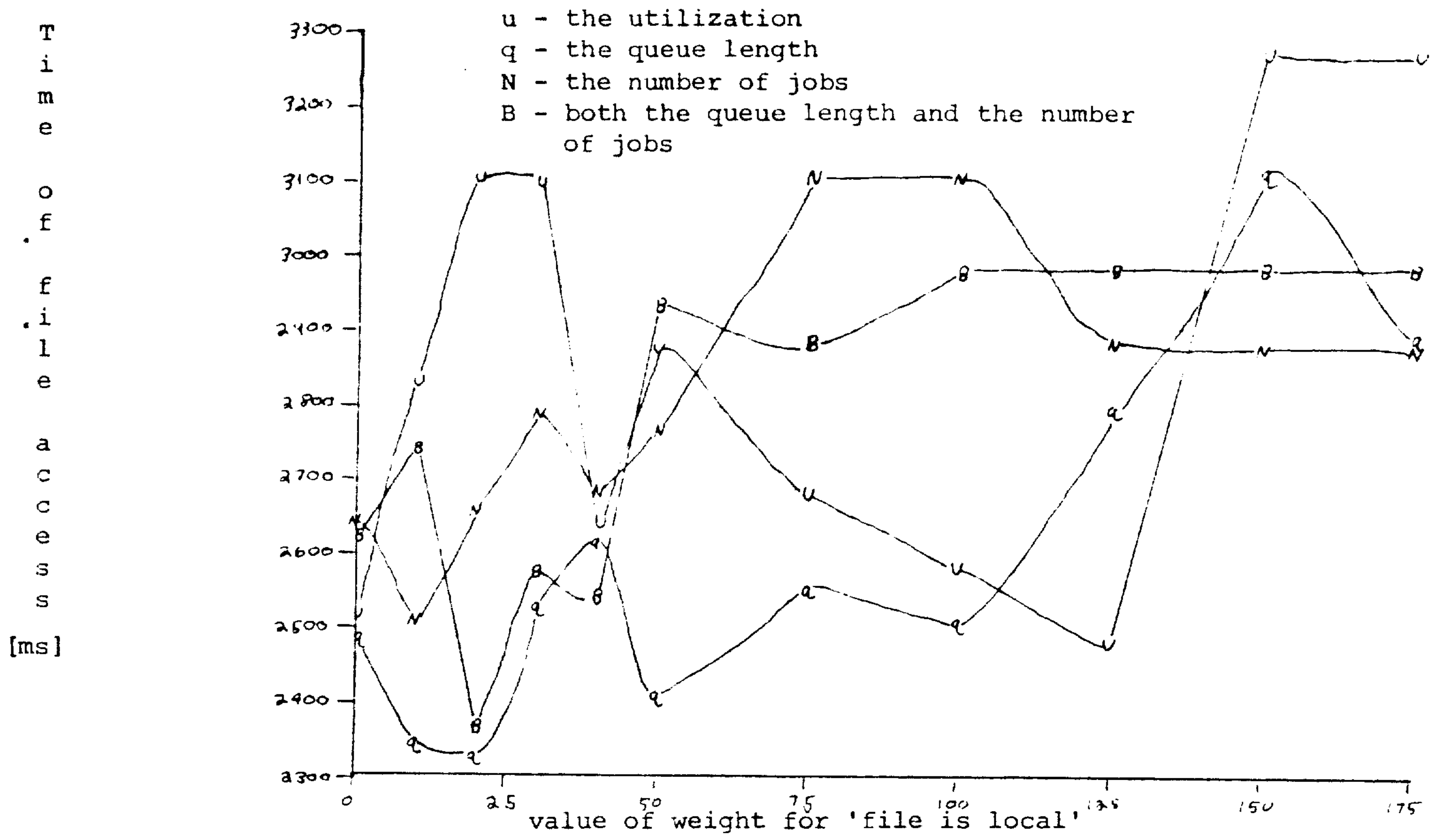


Figure 11. The time of a file access for the read site placement algorithm for workload D.

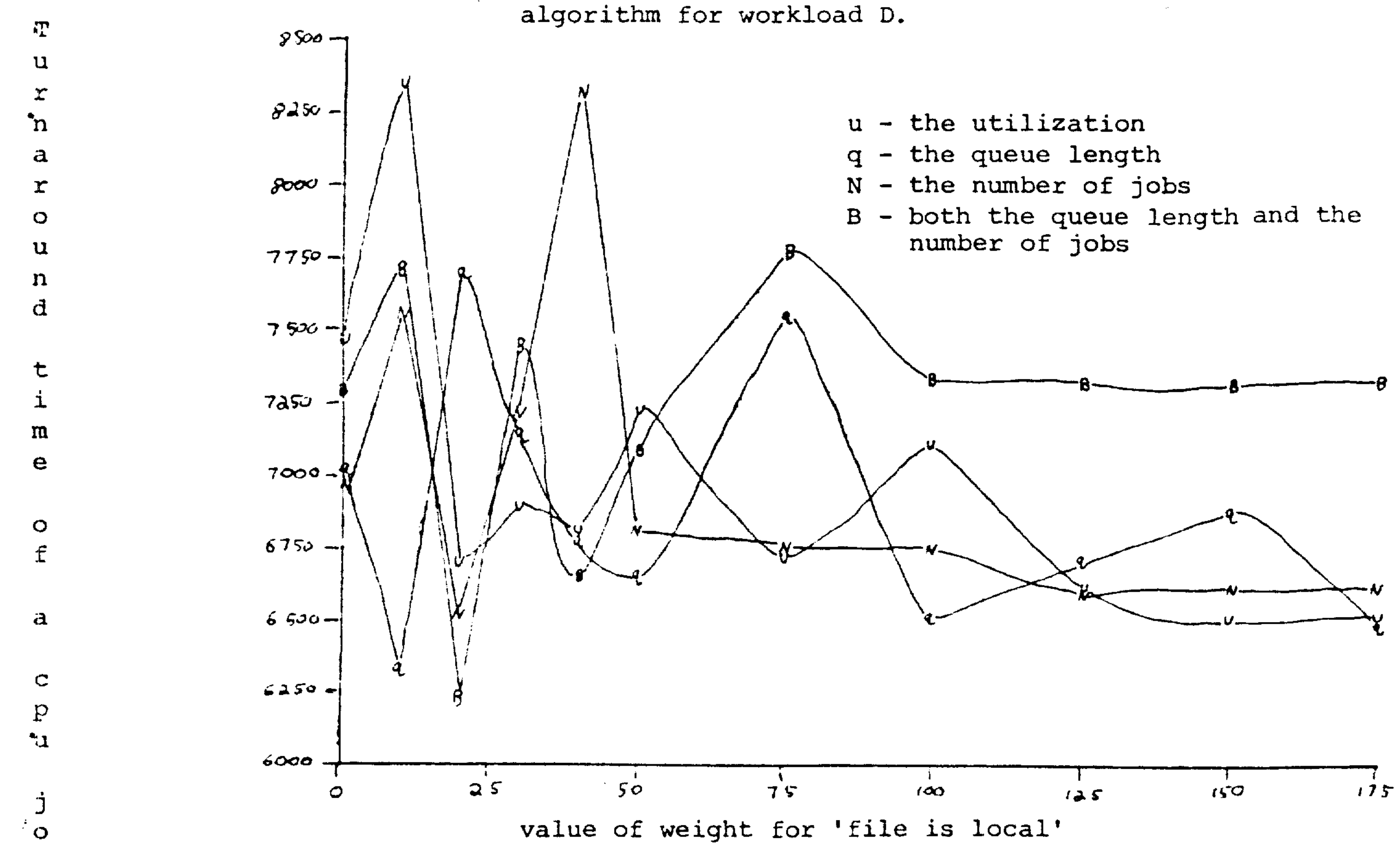
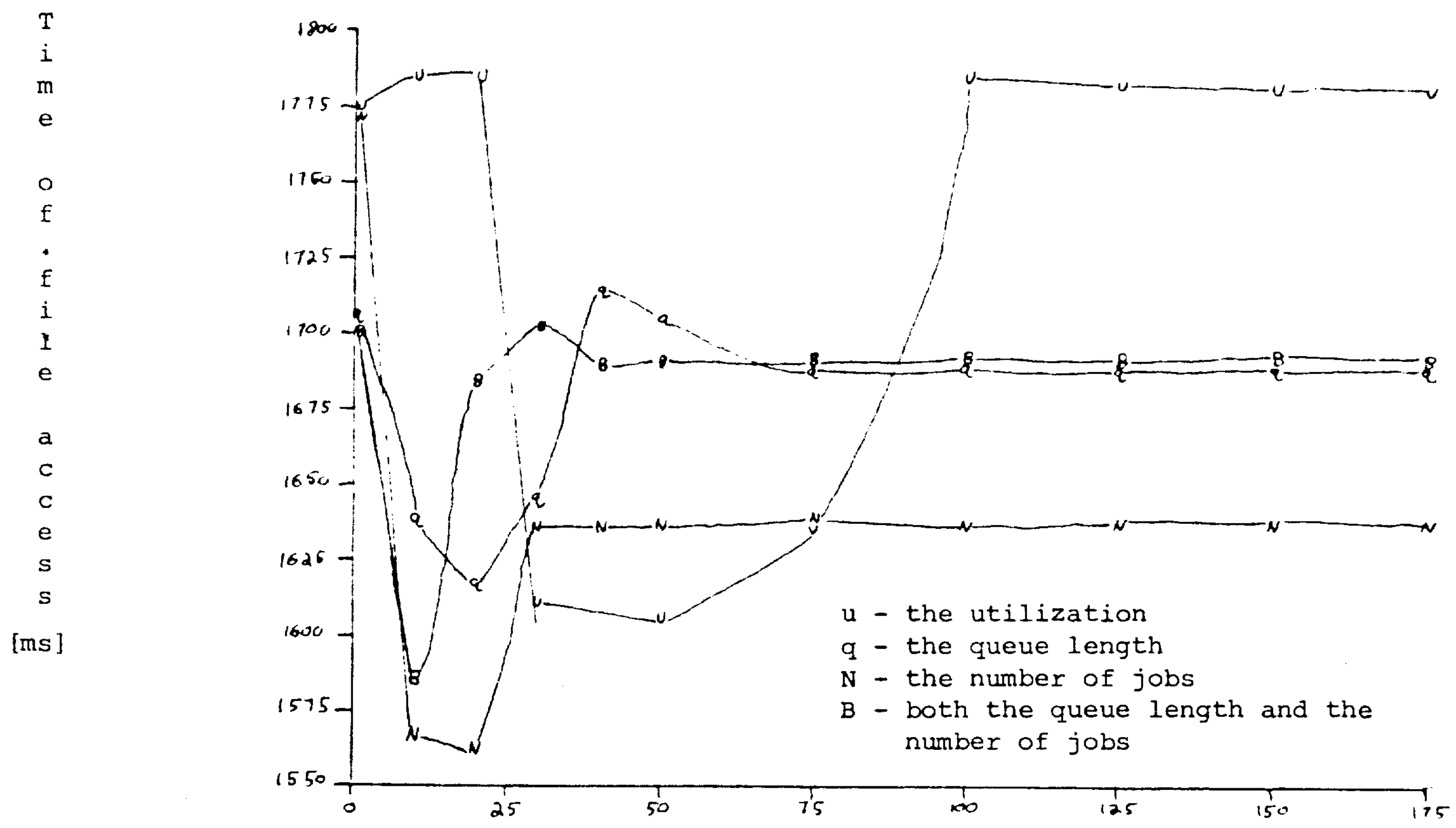
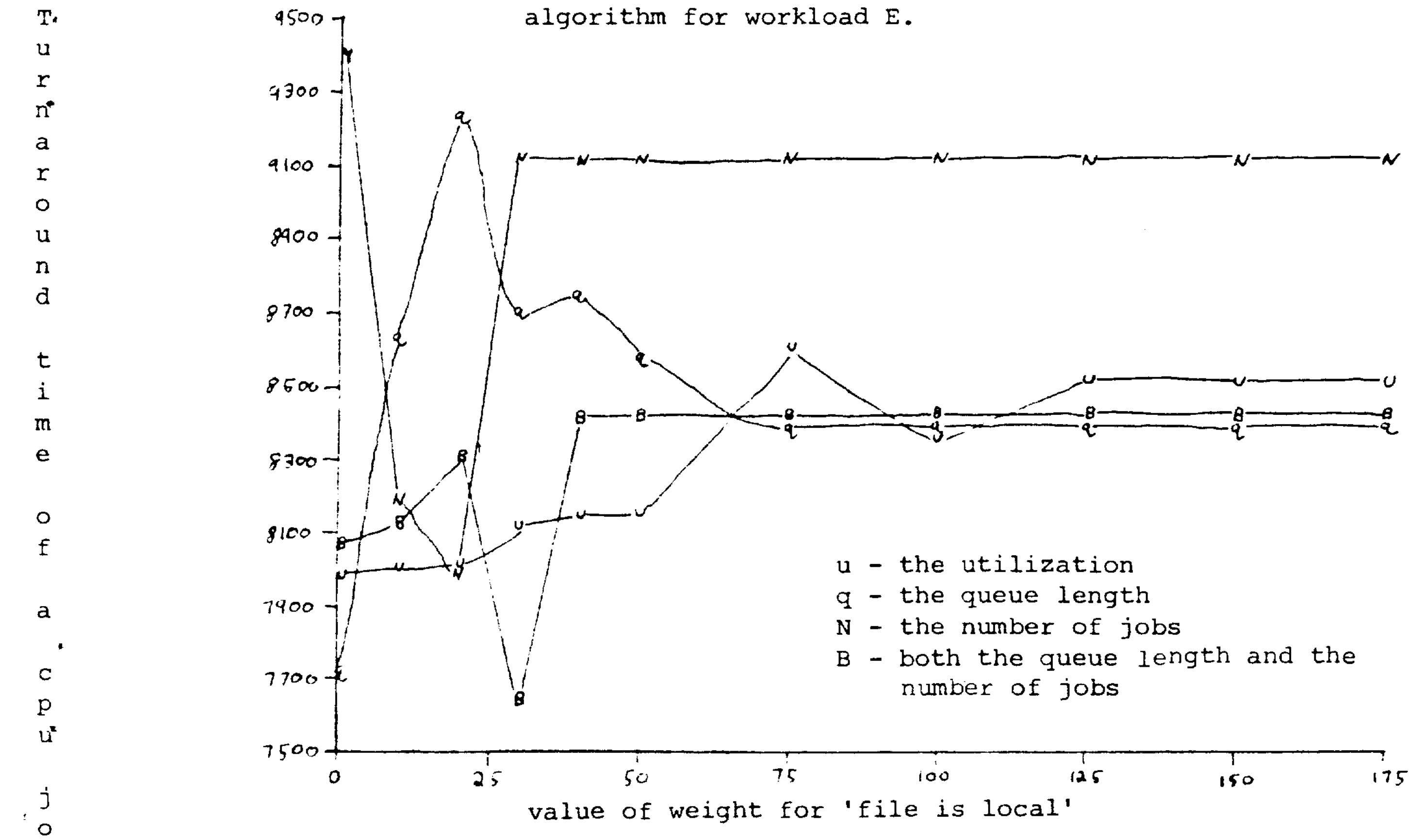


Figure 12. The turnaround time of a cpu-intensive job for the read site placement algorithm for workload D.



value of weight for 'file is local'

Figure 13. The time of a file access for the read site placement algorithm for workload E.



value of weight for 'file is local'

Figure 14. The turnaround time of a cpu-intensive job for the read site placement algorithm for workload E.