Proceedings of the 1985 Winter Simulation Conference
D. Gantz, G. Blais, S. Solomon (eds.)

File Placement and Process Assignment due to Resource Sharing in a Distributed System

Anna Hać

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

ABSTRACT

This paper presents a number of solutions to the problems of data access, concurrency control, transaction serialization, and deadlock control based on the examples of existing file systems. Also, policies for file placement and process assignment are discussed from the viewpoint of their impact on system performance and reliability. The impact of these policies depends on the solutions to the problems referred above. A simulation model of a file system is introduced. This model is validated using data collected in a small business installation. A number of simulation experiments are presented. It is shown how the decision about where to place a file or to assign a process can be based on the workload's and the system's characterization. An algorithm for improving performance and reliability by file placement and/or process assignment is finally presented.

INTRODUCTION

The problems of data access, concurrency control [2], transaction serialization [9], and deadlock control are among the most difficult ones to be solved when designing a file system. The problems of data access are concerned with the unit of data access, and access control. A concurrency control policy allows transaction-type access to shared resources. The unit of concurrency control is usually the same as the unit of data access. The protocols determine the policy for serialization of transactions when accessing the shared resources. The policies for deadlock control allow to resolve a deadlock that may appear in the system. There exist a number of file systems with different solutions to the problems referred above. There are pros and cons to every solution, and not all solutions are independent of each other. Also, depending on the applications, a file system can be designed in several different ways.

The solutions adopted in the following file systems will be referred to in this section:

(1) *XDFS* (XEROX Distributed File System) [31];

(2) *CFS* (Cambridge File Server) [14, 31, 34];

(3) *FELIX* (File Server developed at Bell-Northern Research) [17];

(4) *SWALLOW* (File system developed at MIT) [45];

(5) *CMCFS* (Carnegie-Mellon Central File System) [45];

(6) The file system of *LOCUS* (Distributed operating system based on *UNIX*) [33, 36, 47];

(7) *F-UNIX* (File system developed at Bell Laboratories) [29];

(8) The file system of *DEMOS* (Operating system developed at Los Alamos Scientific Laboratory) [37, 38];

(9) The file system of *MULTICS* (Operating system developed at MIT) [32, 43].

The unit of data access [25, 40] can be a file (*SWALLOW*), a sequential subset of a file (*CFS*, *CMCFS*, *DEMOS*), a page (*FELIX*, *LOCUS*, *MULTICS*), or a subset of a page (*XDFS*). A mechanism for access control is necessary to protect files from unauthorized access. In a capability-based system usually the file identifier (FID) is used as the capability. In an identity-based system, the identity of the user is also required. Systems *CFS*, *FELIX*, *LOCUS* and *SWALLOW* have a capability-based scheme, whereas systems *CMCFS*, *XDFS*, and directory access in *FELIX* are identity-based.

A concurrency control policy [2] allows transaction-type access to shared files. Concurrency control can be handled by the file system (*CFS*, *FELIX*, *LOCUS*, *SWALLOW*, *XDFS*, *DEMOS*) or by the users (*CMCFS*). The unit of concurrency control (i.e., the entity to be locked) is usually a file (*XDFS*, *CFS*, *FELIX*, *SWALLOW*, *CMCFS*); it can also be a page (i.e., a unit of storage allocation) (*CMCFS*, *MULTICS*). The usual concurrency control policy is the single-writer multiple-readers policy, that is:

(a) a transaction can read a file iff the file is not being written;

(b) a transaction can write a file iff the file is not being read or written.

The most popular protocols for transaction serialization are: two-phase locking protocols (*XDFS*, *FELIX*, *CMCFS*) and timestamps based protocols (*SWALLOW*). In the first phase of a two-phase locking protocol, a transaction acquires locks; in the second phase, it releases all locks. A timestamp policy requires that all transactions have the values of their arrival times associated to them and can access files in chronological order of arrival.

A simple example of a deadlock is when two transactions, each having locked a file, require access to the file locked by the other transaction. The policies for deadlock control are [19, 27, 39, 46]:

(a) Deadlock prevention (*FELIX*). A transaction declares all files it needs before starting.

(b) Deadlock detection and resolution (*FELIX*). One of the transactions (or processes) that has caused a deadlock is

selected and preempted, and its locks are released; after a while, that transaction is restarted. The operating system includes a lock manager which detects and resolves deadlocks.

(c) Time-limited locks (*XDFS*). A transaction waits to acquire its locks only for a certain amount of time, and then is timed out; after a while, it retries to acquire its locks. This policy can be acceptable only when the system is not heavily loaded. When the system becomes more congested, then more transactions time out due to their waiting for system resources and locks, rather than because of a system deadlock.

(d) Ordering by timestamps (*SWALLOW*) determines whether a transaction may wait for locks without creating a deadlock. A decision about whether to wait for locks or to abort a transaction is made immediately when a transaction requests access to a file. This decision is made on the basis of the order of timestamps.

(e) None (*CFS*, *CMCFS*). The users choose whether in the case of conflict a request has to be rejected or to be queued. The latter decision may cause a deadlock.

The solutions outlined above influence the design of a distributed file system and the policies for file placement and process assignment. Among the important problems arising in such design are [9, 10, 16, 18]: resource management, performance, reliability, and transparency. These solutions, policies and problems are dependent one on another.

Other actions which may improve performance are file placement and process assignment [1, 4, 8, 28, 30, 42]. A process can be assigned to a machine which is less heavily loaded. Also, a file can be placed on the machine where a transaction which requires access to this file is executed.

This paper investigates the issues of file placement and process assignment in a local area network [11, 16, 26] from the viewpoint of system performance and reliability. A simulation model of a distributed file system is constructed to evaluate our approaches and solutions. Our model of concurrency control represents access to files and sectors, and distributed locks. The model of system workload consists of a number of transaction types. Each type of transaction corresponds to a number of transactions associated with of terminals. Every transaction type is characterized by its demands of the system resources. Due to our interest in file placement and process assignment, special attention is paid to the number of read versus write accesses to each file for every transaction type. Also, the amount of sharing of the files and of the sectors will be considered as an important factor.

In Section 2, we describe the distributed file system to be modeled, and especially its mechanism for concurrency control. Section 3 presents a simulation model of the file system. The results obtained from this model are presented in Section 4. An algorithm for improving performance and reliability is given in Section 5. The algorithm considers workload characterization and system architecture as the important factors of a decision about where to place a file or to assign a process. Finally, Section 6 concludes the paper with a brief description of the problems which remain to be investigated.

THE FILE SYSTEM TO BE MODELED

The file system to be used in our experiments has been designed for a small business installation. This file system will be described from the viewpoint of its solutions to data access, concurrency control, transaction serialization, and

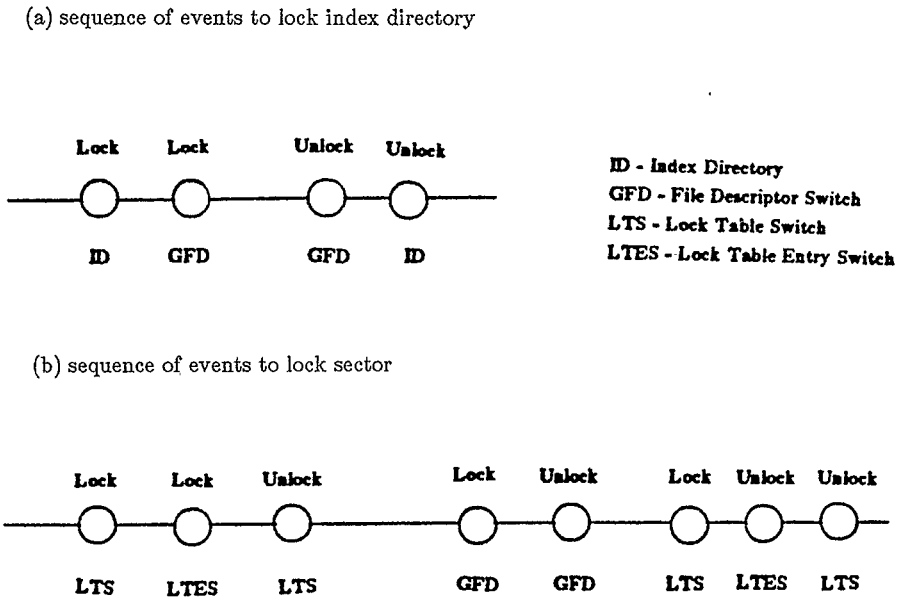

Figure 1. Sequences of events to lock index directory, file descriptor, and sector.

deadlock control. These solutions have significant impact on system performance and reliability, and influence policies for file placement and process assignment.

Our file system is identity based, and the unit of data access is the sector. The file system consists of a set of fixed size blocks. Concurrency control is handled automatically by the file system. The concurrency control policy is the single-writer multiple-readers policy. The unit of concurrency control is the sector. The sequences of events to lock the index directory, a file descriptor and a sector are shown in Fig. 1. Every read or write access to a sector is preceded by an index directory access and a file table switch access. The index directory is locked and accessed to locate the position of a record and possibly to modify the information in a file descriptor. Then, the index directory is unlocked. In order to access a sector, the lock table switch of the file has to be locked to find the entry for the sector, and then to lock the table entry switch of this sector for read and write access. Then the lock table switch of the file is unlocked, and other transactions can lock it for accessing other sectors. A sector can be accessed for reading or writing while its table entry switch is locked. After reading or writing a sector, the file descriptor switch is locked to modify the information, if necessary, and then it is unlocked. The lock table switch of the file is locked again to unlock the lock table entry switch of the sector, and then the lock table switch of the file is unlocked.

This concurrency control policy allows simultaneous, parallel access to different sectors of a shared file. However, every lock requires a certain amount of CPU time to be executed. And the performance degradation due to locking can be significant when the CPU is the system's bottleneck.

There are two policies for controlling deadlocks. The first consists of having transactions declare all the files they will need before starting. This only prevents a deadlock in file descriptor switches. However, this does not prevent a deadlock in the file system, since the unit of concurrency control is the sector. This is why an additional policy is needed which uses time-limited locks. After a certain amount of time, a transaction loses its locks and retries to acquire them again. Several unsuccessful attempts cause a request to be issued to the user to terminate the transaction.

Our description covers only some aspects of this file system. This information is necessary for the purposes of its modeling and analysis. A simulation model of the file system running in a distributed environment is presented in the next section.

A SIMULATION MODEL OF A DISTRIBUTED FILE SYSTEM

The system considered in this study is a distributed file system containing several identical hosts. Each host includes a number of terminals which initiate the processing of transactions. The behavior of a transaction is as follows: each transaction has to be processed in the CPU before it accesses any other service center; it may also be processed by a file disk and by the network server; and it may be delayed because of locks. In its local host, a transaction also produces several display outputs before returning to a user terminal.

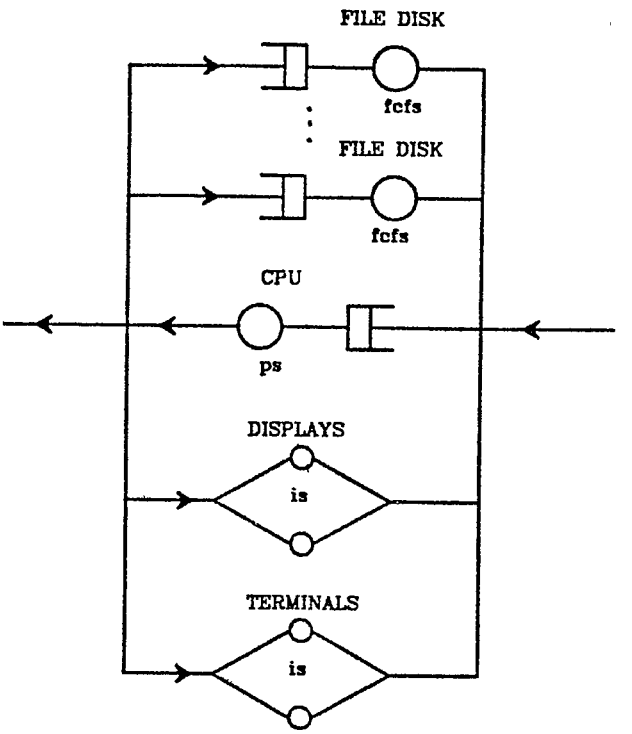

Figure 2. A model of a host.

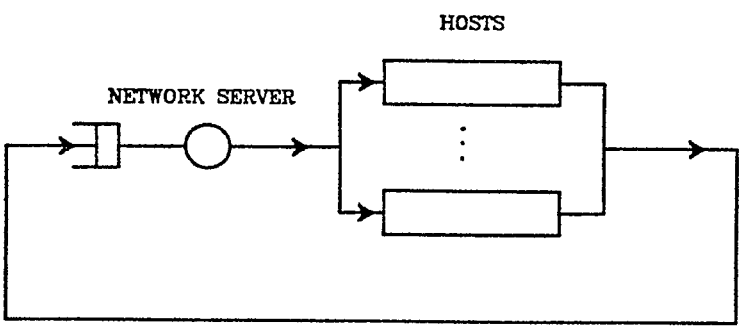

Figure 3. A model of a distributed system.

The model of a single host and that of the multiple host system are shown in Fig. 2 and 3, respectively. The model of each host contains the representation of a number of terminals and display output devices, both modeled as infinite servers (is), a CPU modeled as a processor-sharing (ps) server, and a number of file disks, each disk drive being modeled as a first come first served (fcfs) server. Each file disk contains a number of files. The directory is assumed to be located in main storage.

The workload of the system consists of a number of transactions. It is assumed that each type of transaction defines its own class (i.e., the number of classes is equal to the number of transaction types). Each class of transactions is characterized by the mean resource requirements of each transaction, the sectors of the files locked and unlocked for each disk access for each transaction type; and, for every sector, the mean number of physical disk accesses occurring while the sector is kept locked.

SOME RESULTS

A few simple examples of performance prediction (approached with the RESQ2 queueing network analysis package [41]) in a distributed system using measurements collected in a small business installation will now be discussed to show the applications of the model and to evaluate the performance changes due to resource sharing. The impact of the amount of sharing of the sectors by transaction types on the system's performance will be discussed on an example of a single-host system. This does not detract from the analysis since the amount of sharing is not a peculiar aspect of multiple-host systems. The impact of sharing of distributed locks by transaction types on the system performance will be discussed on an example of a multiple-host system. The time delays due to sharing read and write locks by transaction types will also be presented.

The workload of the system is defined by two types of transactions. The following data are measured for each transaction type:

P - the percentage of the number of transactions of that type in the workload;

I - the number of interactions;

T_{TH} - the mean "think" time of an interaction;

T_S - the mean time of a display output;

T_{CPU} - the mean CPU time consumed by a transaction of that type;

D - the total number of disk I/O operations in a transaction of that type;

S - the total number of display outputs in a transaction of that type;

F - the number of sectors of the files accessed by a transaction of that type;

ND - the number of disk I/O operations done while keeping a sector locked, for every sector.

The following parameter values can be derived from the measurement data:

Average CPU time per interaction $(\frac{T_{CPU}}{I})$

Average number of disk I/O operations per interaction $(\frac{D}{I})$

Average number of display outputs per interaction $(\frac{S}{I})$

Average CPU time between successive disk I/O operations and/or display outputs $(\frac{T_{CPU}}{D+S})$

Probability of accessing the file disk $(\frac{D}{1+D+S})$

Probability of a display output $(\frac{S}{1+D+S})$

The estimated values of the parameters of each transaction type (i.e., of each class in the model) are given in Table I.

The average service times for the following two service centers are the same for all transaction types:

Disk access	$T_{DISK} = 20\ ms$
Network server	$T_{NET} = 0.2\ ms$

Every access to a sector of a file requires four accesses to the CPU due to locking. The average service times of these accesses are:

$T_1^{CPU} = 5.03\ ms$ (Index directory lock and file descriptor lock)

$T_2^{CPU} = 2.60\ ms$ (Sector locking: table switch and table entry switch locks)

Table I. Estimated mean values of the model parameters for a single-host system.

Transaction Type	$T_{CPU}/(D+S)$ [ms]	T_{TH} [ms]	T_S [ms]	D/(1+D+S)	S/(1+D+S)
T1	6	900	15	0.45	0.54
T2	8	900	20	0.495	0.495

Table II. Performance measures obtained from measurement and simulation of a single-host system.

Transaction Types	Number of Transactions	Results	Performance Measures							
			Utilization		Throughput [1/s]			Response Time [s]		
			CPU	DISK	SYSTEM	T1	T2	SYSTEM	T1	T2
T1; T2	2; 1	measurement	0.28	0.12	1.08	0.77	0.32	0.6	0.5	0.8
		simulation	0.29	0.12	1.09	0.78	0.31	0.6	0.5	0.8
T1; T2	2; 2	measurement	0.32	0.12	1.18	0.64	0.55	0.65	0.8	0.5
		simulation	0.33	0.12	1.18	0.64	0.54	0.65	0.8	0.5

$T_3^{CPU} = 0.94\ ms$ (File descriptor lock)

$T_4^{CPU} = 6.91\ ms$ (Sector unlocking: table entry switch and table switch locks)

Note that the average service time for sector unlocking is much higher than that for locking a sector.

The system was modeled using the simulation method. The confidence level of the simulations was 90 percent. The confidence interval width was (-10, +10) percent. Performance measures were calculated for this simulation model. The model was validated with measurements collected in a small business installation. Two simple examples of mixed transactions were run in that installation on a single-host system. The values of performance measures obtained from the simulation results and the measurement results were sufficiently close, as the error was less than 10 percent. The comparison of results obtained from the simulation model and from measurement is presented in Table II. Note that the measurements of system performance were taken on a different hardware installation (i.e., different CPU) than the measurements of the workload, and were used only to validate our model.

Sharing of Sector Locks by Transaction Types

In this example, the performance measures are computed for a single-host system containing one file disk with one file on it. Three sectors of this file are accessed by transaction types. The average number of disk accesses when a sector is kept locked is two. A transaction may be sent to a display or a terminal when it keeps a lock on a sector. This means that no other transaction can access this sector at that time.

In the first example, transactions of type T1 access the first sector only, and transactions of type T2 access the second sector only. The third sector is not accessed, and no sector is shared by transaction types. The results of the experiments are shown in Table III (L1 and L2 in that table represent the table entry switch lock for the first and the second sector, respectively).

Table III. Performance measures for the simulation model when one file is shared by transaction types (no sector is shared).

Transaction Types	Number of Transactions	Performance Measures									
		Utilization				Throughput [1/s]			Response Time [s]		
		CPU	DISK	L1	L2	SYSTEM	T1	T2	SYSTEM	T1	T2
T1; T2	1; 1	0.56	0.25	0.62	0.58	0.56	0.31	0.25	2.68	2.39	3.04
T1; T2	2; 2	0.73	0.33	0.89	0.88	0.70	0.38	0.32	4.79	4.34	5.31
T1; T2	3; 3	0.76	0.34	0.98	0.98	0.73	0.41	0.32	7.29	6.39	8.44
T1; T2	4; 4	0.78	0.35	1.00	1.00	0.70	0.36	0.34	10.53	10.22	10.82

Table IV. Performance measures for the simulation model when one file is shared by transaction types (one sector is shared, P(sharing)=0.1).

Transaction Types	Number of Transactions	Performance Measures										
		Utilization					Throughput [1/s]			Response Time [s]		
		CPU	DISK	L1	L2	L3	SYSTEM	T1	T2	SYSTEM	T1	T2
T1; T2	1; 1	0.56	0.25	0.53	0.53	0.11	0.54	0.29	0.25	2.76	2.52	3.08
T1; T2	2; 2	0.74	0.34	0.83	0.83	0.19	0.70	0.38	0.32	4.81	4.39	5.31
T1; T2	3; 3	0.79	0.36	0.95	0.94	0.24	0.78	0.40	0.38	6.67	6.55	7.00
T1; T2	4; 4	0.81	0.37	0.99	0.99	0.25	0.80	0.42	0.38	9.16	8.67	9.71

Table V. Performance measures for the simulation model when one file is shared by transaction types (one sector is shared, P(sharing)=0.3).

Transaction Types	Number of Transactions	Performance Measures										
		Utilization					Throughput [1/s]			Response Time [s]		
		CPU	DISK	L1	L2	L3	SYSTEM	T1	T2	SYSTEM	T1	T2
T1; T2	1; 1	0.54	0.25	0.41	0.39	0.34	0.50	0.28	0.22	3.08	2.68	3.59
T1; T2	2; 2	0.71	0.32	0.63	0.63	0.56	0.71	0.40	0.32	4.73	4.13	5.34
T1; T2	3; 3	0.80	0.36	0.79	0.79	0.69	0.78	0.42	0.36	6.82	6.26	7.47
T1; T2	4; 4	0.83	0.37	0.86	0.86	0.76	0.80	0.42	0.38	9.11	8.64	9.63

In the second example, transactions of type T1 access the first and the third sector, and transactions of type T2 access the second and the third sector. The third sector is shared by transaction types. The results of our experiments are shown in Tables IV, V, VI, VII and VIII for different amounts of sharing (the symbols L1, L2, and L3 represent the table entry switch lock for the first, second and third sector, respectively; P(sharing) is the probability of accessing the third sector by transactions of type T1 and T2, and 1-P(sharing) is the probability of accessing the first or the second sector by transactions of type T1 or T2, respectively).

The comparison of results show that the table entry switch lock (L1, L2 or L3) of a sector may become the limiting component in the system. The reason for this is that a transaction can lock a sector and then be sent to a display or to a terminal where the time spent is very long in comparison with the time spent at the other service centers. A table entry switch lock of a sector may become a bottleneck even if no sector is shared (Table III). The amount of sharing is also an important factor. The simulation results show that the performance degradation is crucial when the probability of sharing is 0.5 or higher. A comparison of the results presented in Tables III, IV, V, VI, VII and VIII shows that the bottleneck changes from the table entry switch lock of non-shared sectors to the table entry switch lock of the shared sector, when the probability of sharing becomes equal to 0.5. When the probability of sharing is equal to 0.1 or 0.3 (Tables IV and V), the performance is slightly better than in the no-sharing case (Table III). The reason is that, in this case, the system becomes saturated faster when two sectors of a file are accessed rather than three sectors. When the numbers of accesses to the sectors are balanced (probability of sharing equal to 0.3, Table V), more transactions may be executed in the system without creating a bottleneck. When the probability of sharing is equal to 0.5, 0.7 or 0.9 (Tables VI, VII and VIII), the table entry switch lock of the shared sector becomes the limiting system component. The degradation of performance is significant when the numbers of accesses to the sectors are unbalanced (probability of sharing

Table VI. Performance measures for the simulation model when one file is shared by transaction types (one sector is shared, P(sharing)=0.5).

Transaction Types	Number of Transactions	Performance Measures										
		Utilization					Throughput [1/s]			Response Time [s]		
		CPU	DISK	L1	L2	L3	SYSTEM	T1	T2	SYSTEM	T1	T2
T1; T2	1; 1	0.53	0.24	0.29	0.29	0.53	0.47	0.25	0.22	3.36	3.16	3.59
T1; T2	2; 2	0.66	0.29	0.43	0.43	0.81	0.62	0.35	0.27	5.48	4.77	6.41
T1; T2	3; 3	0.70	0.32	0.49	0.48	0.94	0.66	0.35	0.31	8.21	7.66	8.81
T1; T2	4; 4	0.74	0.33	0.52	0.51	0.97	0.64	0.33	0.30	11.61	11.24	12.40

Table VII. Performance measures for the simulation model when one file is shared by transaction types (one sector is shared, P(sharing)=0.7).

Transaction Types	Number of Transactions	Performance Measures										
		Utilization					Throughput [1/s]			Response Time [s]		
		CPU	DISK	L1	L2	L3	SYSTEM	T1	T2	SYSTEM	T1	T2
T1; T2	1; 1	0.50	0.22	0.16	0.18	0.67	0.47	0.25	0.22	3.38	3.12	3.68
T1; T2	2; 2	0.60	0.27	0.23	0.22	0.94	0.56	0.29	0.27	6.27	5.97	6.59
T1; T2	3; 3	0.61	0.27	0.23	0.24	0.99	0.59	0.31	0.28	9.29	8.77	9.86
T1; T2	4; 4	0.63	0.28	0.25	0.25	1.00	0.61	0.32	0.29	12.18	11.56	12.90

Table VIII. Performance measures for the simulation model when one file is shared by transaction types (one sector is shared, P(sharing)=0.9).

Transaction Types	Number of Transactions	Performance Measures										
		Utilization					Throughput [1/s]			Response Time [s]		
		CPU	DISK	L1	L2	L3	SYSTEM	T1	T2	SYSTEM	T1	T2
T1; T2	1; 1	0.46	0.21	0.05	0.05	0.82	0.43	0.23	0.21	3.77	3.48	3.86
T1; T2	2; 2	0.53	0.24	0.06	0.07	0.99	0.49	0.27	0.22	3.22	2.81	3.73
T1; T2	3; 3	0.52	0.24	0.07	0.06	1.00	0.50	0.27	0.24	7.10	6.48	7.50
T1; T2	4; 4	0.52	0.24	0.06	0.07	1.00	0.53	0.27	0.26	14.21	13.93	14.50

equal to 0.7 or 0.9, Tables VII and VIII). In this case, the table entry switch lock of the third sector is the limiting system component, and the delay due to waiting for this lock causes significant degradation of system performance.

These results show that many factors have an impact on system performance when sectors are shared by various transaction types. The addition of a shared sector may increase or decrease system performance depending on the probability of sharing. The results presented above were obtained when the average number of disk accesses when a sector is kept locked was equal to two. Also, a transaction could be sent to a display or a terminal while keeping a lock on a sector. The time spent at a display or a terminal is very long in comparison with the time spent at the other service centers. This is why the table entry switch locks could become system bottlenecks in these experiments. For the same number of disk accesses made when a sector is kept locked, but when a transaction cannot be sent to a display or a terminal while keeping a lock of a sector, the table entry switch locks may not become system bottlenecks. Note that the number of disk accesses made when a sector is kept locked is also an important factor [22]. When this number increases, the lock of a sector is kept longer, and a bottleneck effect similar to that of a table entry switch lock may be observed.

It is important to know the reasons which cause bottlenecks on locks, since these reasons are not exactly the same as the reasons which cause bottlenecks on a system's service centers. Besides the workload's demands on system resources, the important factors to be considered are the number of disk accesses made when a sector is kept locked, and the amount of sharing for every sector.

Sharing of Distributed Locks by Transaction Types

In this example, the workload of the system is defined by five types of transactions.

The estimated values of the parameters of each transaction type (i.e., each class in the model) are given in Tables IX and X for a single-host system and for a system with two hosts, respectively. In Table X only the values of parameters different from those in Table IX are shown. Also, the probability of accessing a remote host (P_H) is derived assuming that a number of the disk I/O operations for each transaction type is performed in the remote host. The probability of accessing a file disk (P_F) by a remote transaction is calculated on the basis of the number of local accesses made by this transaction. Note that the remote transactions do not do remote display outputs and do not return to a user terminal in the remote host.

The average service times for the following two service centers are the same for all transaction types:

Disk access	$T_{DISK} = 30\ ms$
Network server	$T_{NET} = 0.2\ ms$

The performance measures are computed for a distributed system containing two identical hosts. Each host includes one file disk with three files on it. Performance measures were calculated for the analytic model [22] in the no-sharing case, and when one file is shared. The probability of locking delay is estimated as the ratio of the number of disk accesses occurring while keeping the file locked to the total number of disk accesses of a transaction of class i. The results of the comparison of performance measures are presented in Tables XI and XII.

The results presented in Table XI show that, when the CPU is the bottleneck in the system, then changing the probability of delay does not have any significant impact on the system performance.

The results presented in Table XII show that even when the CPU is the bottleneck in a system in which a file is shared by transactions of different types, then changing the probability of locking delay may have a significant impact on the system's performance. In this case, when the probability of delay increases, then the CPU is no longer a bottleneck in the system.

The comparison of the results from the models shows that the performance degradation due to locking can be significant. This degradation is higher when files are shared by different transaction types. This is why the granularity of locking is very important for system performance. For instance, if locks are on records or sectors rather than on files, then the likehood of sharing is smaller and the performance can be higher. Also the number of disk I/O operations done while keeping a file locked has a significant impact on performance.

Table IX. Estimated values of the model parameters for a single-host system.

Transaction Type	$T_{CPU}/(D+S)$ [ms]	T_{TH} [ms]	T_S [ms]	D/(1+D+S)	S/(1+D+S)
T1	42	1000	15	0.4	0.49
T2	35	10000	25	0.8	0.19
T3	224	1000	20	0.75	0.2
T4	32	500	15	0.966	0.027
T5	15	2000	15	0.3	0.61

Table X. Estimated values of the model parameters for a system of two hosts.

Transaction Type	D/(1+D+S+H)	S/(1+D+S+H)	P_H	P_F
T1	0.39	0.47	0.04	0.79
T2	0.74	0.17	0.07	0.987
T3	0.7	0.19	0.07	0.943
T4	0.88	0.025	0.088	0.993
T5	0.29	0.59	0.03	0.778

Table XI. Performance measures for the distributed system model when no file is shared by transaction types.

Host Number	Transaction Types	Number of Transactions of Each Class	Probability of Locking Delay	Performance Measures			
				Utilization		Throughput [1/s]	Response Time [s]
				CPU	Disk		
1	T1; T2; T5	2; 2; 2	0.1	0.38	0.22	0.59	7.76
2	T3; T4; T5	2; 2; 2	0.1	0.99	0.45	1.18	3.24
1	T1; T2; T5	2; 2; 2	0.9	0.37	0.22	0.58	7.9
2	T3; T4; T5	2; 2; 2	0.9	0.98	0.44	1.17	3.3

Table XII. Performance measures for the distributed system model when one file is shared by transaction types.

Host Number	Transaction Types	Number of Transactions of Each Class	Probability of Locking Delay	Performance Measures			
				Utilization		Throughput [1/s]	Response Time [s]
				CPU	Disk		
1	T1; T2; T5	2; 2; 2	0.1	0.38	0.22	0.59	7.78
2	T3; T4; T5	2; 2; 2	0.1	0.99	0.45	1.18	3.24
1	T1; T2; T5	2; 2; 2	0.25	0.37	0.22	0.57	8.12
2	T3; T4; T5	2; 2; 2	0.25	0.96	0.43	1.15	3.37
1	T1; T2; T5	2; 2; 2	0.5	0.30	0.18	0.46	10.63
2	T3; T4; T5	2; 2; 2	0.5	0.78	0.35	0.93	4.60
1	T1; T2; T5	2; 2; 2	0.75	0.21	0.13	0.33	15.76
2	T3; T4; T5	2; 2; 2	0.75	0.56	0.25	0.67	7.12
1	T1; T2; T5	2; 2; 2	0.9	0.18	0.11	0.28	19.04
2	T3; T4; T5	2; 2; 2	0.9	0.47	0.21	0.56	8.88

Sharing of Read and Write Locks by Transaction Types

Three simple examples of time delays due to locking a file show the delay difference due to read and write accesses for various lock granularities.

In this example, the workload of the system is defined by types of transactions given in Table IX.

The system was modeled using simulation and numerical methods [23]. The confidence level of the simulation was 90 percent. The confidence interval widths was (-10; +10) percent. The experiments were performed with various lock granularities. Locks were assumed to be on 1 sector, 2 sectors or 3 sectors. The time delays due to locking for various lock granularities are presented in Tables XIII, XIV and XV for the simulation model.

The results presented here show that read and write accesses cause different locking delays. These delays also depend on the granularity of locking.

AN ALGORITHM FOR FILE PLACEMENT AND PROCESS ASSIGNMENT

The policies for improving performance and reliability through file placement and process assignment depend on the system solutions to the problems of data access, concurrency control, transaction serialization, and deadlock control.

In our example file system, the problems of data access, transaction serialization, and deadlock control depend on the concurrency control policy. This concurrency control mechanism is a heavy consumer of processor time, and therefore, very expensive.

The important factor to be considered is the workload (classes of transactions and their demands on the system resources), and in particular the number of read and write accesses to each file for every transaction type, and the amount of sharing by transaction types for every sector.

Let us define a number of parameters, in addition to those listed in Section 4. These additional parameters are:

N - the number of transaction types in the system;

n_i - the number of transactions of type i;

pr_i^f - the unconditional probability of read access to file f by a transaction of type i;

pw_i^f - the unconditional probability of write access to file f by a transaction of type i;

ps_i^{sf} - the unconditional probability of read/write access to a shared sector s of file f by a transaction of type i;

k - the parameter which affects the considered ratio of the total number of read and write accesses in the system.

Let $k=2$.

The value of k is based on the experiments of the presented system. These experiments showed that the critical value of the probabilities of sharing and writing is equal to 0.5. In our algorithm, we consider the ratio of the total number of read and write accesses in the system greater than

Table XIII. Time delays due to locking of 1 sector.

Transaction Types	Number of Transactions of Each Class	Time Delays [ms]	
		Read Lock	Write Lock
T1; T2	1; 1	0.9	2.4
T1; T2	2; 2	4.16	7.8
T1; T2	4; 4	16.9	18.5

Table XIV. Time delays due to locking of 2 sectors.

Transaction Types	Number of Transactions of Each Class	Time Delays [ms]	
		Read Lock	Write Lock
T1; T2	1; 1	8.3	9.95
T1; T2	2; 2	12.1	17.6
T1; T2	4; 4	52.1	57.56

Table XV. Time delays due to locking of 3 sectors.

Transaction Types	Number of Transactions of Each Class	Time Delays [ms]	
		Read Lock	Write Lock
T1; T2	1; 1	9.5	15.2
T1; T2	2; 2	42.6	51
T1; T2	4; 4	57.6	69.3

k:1 ($k=2$) to make sure that we can apply policies for file placement without causing degradation of system performance. When the ratio of the total number of read and write accesses in the system is smaller than 1:k ($k=2$), we apply only the policies for process assignment. Note that the value k may be different for different systems, and for different number of hosts. However, in the experiments presented, the numbers of files, sectors and hosts were small, and we expect that, when these numbers increase, the probability of sharing and writing may be less critical, and the value k may be decreased even to $k=1$. When the ratio of the total number of read and write accesses in the system varies between those two (i.e., 1:k and k:1), we consider the amount of sharing as an important factor. If the ratio of the total number of shared read and shared write accesses in the system is smaller than 1:k or greater than k:1, then we apply the appropriate policy; if it does not, then we cannot predict if the use of any policy will increase or decrease performance.

The performance measures considered in our approach are the utilization of each service center, the system's throughput and mean response time, and the throughput and mean response time for each class of transactions. The reliability measures are the mean time to failure and system reliability.

Our algorithm for improving performance and reliability has to be applied to every file in the system. It uses measurable parameters only, and can be applied to any file system whose workload can be characterized using the parameters introduced in this paper. The detection of bottlenecks on each host may be done using the approach given in [20]. That approach detects bottlenecks on the basis of the workload demands on system resources, using measurable parameters. A bottleneck on a lock is detected in the same way, but the mean time spent in all service centers when a lock is kept must be considered. The performance measures may be calculated using [20, 22, 23]. The mean time to failure and system reliability may be calculated using [21].

The approaches given in [20, 21, 22, 23] use measurable parameters only. The performance and reliability measures may also be calculated using, for instance, simulation techniques.

The algorithm presented here has been constructed for system designers, but could also be invoked when a file or a process is created. This algorithm can be used statically to decide of file placement, and static process assignment. It may also be used dynamically when a prediction of the future behavior of processes in the system is possible. This prediction may be based on the past behavior of processes in short period of time, assuming that workload demands on system resources remain the same. Also, performance and reliability measures may be detected, and any substantial change in their values may invoke this algorithm. The dependence of the system solutions to the problems of data access, concurrency control, transaction serialization, and deadlock control, is characterized by the values of the measurable parameters used in this algorithm. This is why

the algorithm can be applied to any of the file systems reviewed in Section 1. Note that the algorithm is invoked for every file in the system, and a decision concerning file placement or assignment of the process which accesses this file is made for each file separately. The decision of process assignment may be made on the basis of process demands on system resources. Another solution is to predict system performance and reliability for every process accessing the file, and to assign the process to a host to maximize performance and reliability. Since this algorithm is invoked iteratively, in the next step another process may be assigned, but also the other decision may be made (for instance, placement of a file). The other problems arise when a process accesses several files. In this case the process assignment may increase the number of remote accesses, and decrease performance and reliability. The decision which process is to be assigned takes into account the number of files it accesses, and the number of read and write accesses. And as in the previous case, the other solution is to predict system performance and reliability for every process accessing the file, and to choose the process that assignment maximizes performance and reliability.

The algorithm keeps on its stack the last two solutions, and requires a number of iterations. The solution which is chosen is the last before system performance and reliability started decreasing.

Algorithm for every file f :

begin

Initial state: the worst performance and reliability (i.e., zero throughput, infinite response time, zero mean time to failure).

A : Detect the bottleneck BK_h on each host h, and the system bottleneck BK_s.

$$\text{if } \sum_{i=1}^{N} n_i\, pr_i^f > k \sum_{i=1}^{N} n_i\, pw_i^f \text{ then } B$$

$$\text{else}$$

$$\text{if } k \sum_{i=1}^{N} n_i\, pr_i^f < \sum_{i=1}^{N} n_i\, pw_i^f$$

$$\text{and } BK_s <> \text{network server then } C$$

$$\text{else } D$$

B: File may be placed on any host where this file is to be accessed

$$\text{if } BK_s = \text{network server then } E$$

$$\text{else } C$$

C:

for $h=1$, *number_of_hosts* do

for $H=1$, *number_of_hosts* do

if BK_h *utilization on host* h

$< k * BK_H$ *utilization on host* H

and

$k *$ *network server utilization*

$< BK_h$ *utilization on host* h *then*

File f has to be moved from host H to host h

else

if BK_h *utilization on host* h

$> k * BK_H$ *utilization on host* H

and

$k *$ *network server utilization*

$< BK_H$ *utilization on host* H *then*

The process which accesses file f and that moving of this process maximizes performance and reliability, is moved from host h to host H

goto E

D:

$$\text{if } \sum_{i=1}^{N} ps_i^{sf} n_i\, pr_i^f > k \sum_{i=1}^{N} ps_i^{sf} n_i\, pw_i^f \text{ then } B$$

$$\text{else}$$

$$\text{if } k \sum_{i=1}^{N} ps_i^{sf} n_i\, pr_i^f < \sum_{i=1}^{N} ps_i^{sf} n_i\, pw_i^f$$

$$\text{and } BK_s <> \text{network server then } C$$

$$\text{else } E$$

E: Calculate performance and reliability measures. Compare with performance and reliability measures obtained in the previous step.

if *performance and/or reliability increase then goto* A

else

choose the previous solution

end.

This algorithm provides a general policy for performance and reliability improvement through file placement and process assignment. A number of files to be placed or a number of processes to be assigned may be obtained by using this algorithm iteratively.

CONCLUSION

The approach presented in this paper uses policies for improving performance and reliability through file placement and process assignment depend on the system solutions to data access, concurrency control, transaction serialization, and deadlock control.

An example file system with its solutions to the problems referred above was modeled, and the model was validated using measurement data collected in a small business installation. A number of simulation experiments were executed to evaluate system performance for different amounts of sharing of the sectors by transaction types, as well as sharing distributed locks by transaction types, and sharing read and write locks.

An algorithm was also presented which provides a general policy for performance and reliability improvements. This algorithm uses measurable parameters only, and bases its decisions about where to place a file or to assign a process on the number of read and write accesses, the number of transactions, the amount of sector sharing, and the utilization of the bottleneck of each host.

This algorithm can be applied to any of the file systems reviewed in Section 1; however, more specific policies will depend on the system architecture and the solutions to data access, concurrency control, transaction serialization, and deadlock control. These solutions are not independent of each other.

ACKNOWLEDGMENTS

The author wishes to thank Praful Shah for many enlightening discussions on the presented file system and its concurrency control mechanism, and for providing the data used in this paper.

REFERENCES

[1] A. K. Agrawala and S. K. Tripathi, Models of Load Sharing Policies in Distributed Systems, Proc. Decision and Control Conference (Dec. 1981, San Diego, California).

[2] P. A. Bernstein and N. Goodman, Concurrency Control in Distributed Database Systems, Computing Surveys, 13, 2 (1981) 186-221.

[3] H. Besharatian, A Methodology for the Design of Reliable Communication Networks in Distributed Processing Systems, Proc. Eighth Data Communication Symposium, ACM SIGCOMM Computer Communication Review, Vol. 13, No. 4 (Oct. 1983) 131-140.

[4] A. D. Birrell, R. Levin, R. M. Needham and M. D. Schroeder, Grapevine: An Exercise in Distributed Computing, Proc. Eighth ACM Symposium on Operating Systems Principles (Dec. 1981) 178-179.

[5] A. D. Birrell and B. J. Nelson, Implementing Remote Procedure Calls, Proc. Ninth ACM Symposium on Operating Systems Principles (October 1983) 3.

[6] A. Borg, J. Baumbach and S. Glazer, A Message System Supporting Fault Tolerance, Proc. Ninth ACM Symposium on Operating Systems Principles (October 1983) 90-99.

[7] P. Brereton, Detection and Resolution of Inconsistencies among Distributed Replicates of Files, ACM Operating Systems Review (1982) 10-13.

[8] L. Casey and N. Shelness, A Domain Structure for Distributed Computer Systems, Proc. Sixth ACM Symposium on Operating Systems Principles (Nov. 1977) 101-108.

[9] R. Cheng and J. W. S. Liu, Reliable Synchronization in Computer Networks, Proc. 13th IEEE International Symposium on Fault-Tolerant Computing (1983) 178-181.

[10] D. R. Cheriton and W. Zwaenepoel, The Distributed V Kernel and its Performance for Diskless Workstations, Proc. Ninth ACM Symposium on Operating Systems Principles (October 1983) 129-140.

[11] D. D. Clark, K. T. Pogran and D. P. Reed, An Introduction to Local Area Network, Proc. IEEE, 66, 11 (November 1978) 1497-1517.

[12] E. D. Coffman, Jr., E. Gelenbe and B. Plateau, Optimization of the Number of Copies in a Distributed Data Base, IEEE Transactions on Software Engineering, SE-7, 1 (1981) 78-84.

[13] D. H. Craft, Resource Management in a Decentralized System, Proc. Ninth ACM Symposium on Operating Systems Principles (October 1983) 11-19.

[14] J. Dion, The Cambridge File Server, ACM Operating Systems Review (Oct. 1980) 26-35.

[15] C. A. Ellis, Consistency and Correctness of Duplicate Database Systems, Proc. Sixth ACM Symposium on Operating Systems Principles (Nov. 1977) 67-84.

[16] D. Ferrari and T. P. Lee, Modeling File System Organizations in a Local Area Network Environment, Proc. IEEE Conference of Computer Data Engineering (April 1984); also, Report No. UCB/CSD 83/142 (October 1983) Computer Science Division (EECS) University of California, Berkeley, California 94720.

[17] M. Fridrich and W. Older, The FELIX File Server, Proc. Eighth ACM Symposium on Operating Systems Principles (Dec. 1981) 37-44.

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

[19] J. Gray, Notes on Data Base Operating Systems, IBM Research Report, RJ2188(30001) (Feb. 23, 1978) IBM Research Laboratory, San Jose, California 95193.

[20] A. Hać, Computer System and Workload Parametric Model with Measured Results, in: Parallel and Large Scale Computers: Performance, Architecture, Applications (M. Ruschitzka et al., ed.) IMACS/North-Holland, Amsterdam (1983) 21-30.

[21] A. Hać, A System Reliability Model with Classes of Failures, IEEE Transactions on Reliability, Vol. R-34, No. 1 (1985).

[22] A. Hać, An Approximation of Delays Due to Locking of Files in a Distributed System, Proc. Eight IEEE International Computer Software and Applications Conference COMPSAC 84 (Nov. 7-9, 1984, Chicago, Illinois) 58-67.

[23] A. Hać, On the Modeling of Read and Write Locks in a Distributed System, Proc. 13th Annual ACM Computer Science Conference (March 12-14, 1985, New Orleans, Louisiana) 373-380.

[24] E. Holler, Multiple Copy Update, Distributed Systems - Architecture and Implementation, Lecture Notes in Computer Science 105, Springer-Verlag (1981) 284-303.

[25] B. W. Lampson, Atomic Transactions, Distributed Systems - Architecture and Implementation, Lecture Notes in Computer Science 105, Springer-Verlag (1981) 246-264.

[26] B. W. Lampson, Ethernet, Pub and Violet, Distributed Systems - Architecture and Implementation, Lecture Notes in Computer Science 105, Springer-Verlag (1981) 446-484.

[27] G. LeLann, Synchronization, Distributed Systems - Architecture and Implementation, Lecture Notes in Computer Science 105, Springer-Verlag (1981) 266-282.

[28] 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.

[29] G. W. R. Luderer, H. Che, J. P. Haggerty, P. A. Kirslis and W. T. Marshall, A Distributed UNIX System Based on a Virtual Circuit Switch, Proc. Eighth ACM Symposium on Operating Systems Principles (Dec. 1981) 160-168.

[30] G. Mcdaniel, METRIC: a Kernel Instrumentation System for Distributed Environments, Proc. Sixth ACM Symposium on Operating Systems Principles (Nov. 1977) 93-99.

[31] J. G. Mitchell and J. Dion, A Comparison of Two Network-Based File Servers, Comm. ACM, 25, 4 (April 1982) 233-245.

[32] W. A. Montgomery, Measurements of Sharing in MULTICS, Proc. Sixth ACM Symposium on Operating Systems Principles (Nov. 1977) 85-90.

[33] E. T. Mueller, J. D. Moore and G. J. Popek, A Nested Transaction Mechanism for LOCUS, Proc. Ninth ACM Symposium on Operating Systems Principles (October 1983) 71-89.

[34] R. M. Needham and A. J. Herbert, The Cambridge Distributed Computing System, Addison-Wesley (Reading, Mass., 1982).

[35] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. Walker, E. Walton, J. Chow, D. Edwards, S. Kiser and C. Kline, Detection of Mutual Inconsistency in Distributed Systems, IEEE Transactions on Software Engineering, SE-9, 3 (1983) 240-247.

[36] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin and G. Thiel, LOCUS, A Network Transparent, High Reliability Distributed System, Proc. Eighth ACM Symposium on Operating Systems Principles (Dec. 1981) 169-177.

[37] M. L. Powell, The DEMOS File System, Proc. Sixth ACM Symposium on Operating Systems Principles (Nov. 1977) 33-42.

[38] M. L. Powell and B. P. Miller, Process Migration in DEMOS/MP, Proc. Ninth ACM Symposium on Operating Systems Principles (October 1983) 110-119.

[39] D. P. Reed and R. K. Kanodia, Synchronization with Eventcounts and Sequencers, Proc. Sixth ACM Symposium on Operating Systems Principles (Nov. 1977) 91.

[40] M. Satyanarayanan, A Study of File Sizes and Functional Lifetimes, Proc. Eighth ACM Symposium on Operating Systems Principles (Dec. 1981) 96-108.

[41] C. H. Sauer, E. A. MacNair and J. F. Kurose, The Research Queueing Package, Version 2: CMS Users Guide, IBM Research Report RA 139, 41127 (April 1982).

[42] M. D. Schroeder, A. D. Birrell and R. M. Needham, Experience with Grapevine: the Growth of a Distributed System, Proc. Ninth ACM Symposium on Operating Systems Principles (October 1983) 141-142.

[43] M. D. Schroeder, D. D. Clark and J. H. Saltzer, The MULTICS Kernel Design Project, Proc. Sixth ACM Symposium on Operating Systems Principles (Nov. 1977) 43-56.

[44] A. Z. Spector, Performing Remote Operations Efficiently on a Local Computer Network, Proc. Eighth ACM Symposium on Operating Systems Principles (Dec. 1981) 76-77.

[45] L. Svobodova, File Servers for Network-Based Distributed Systems, IBM Tech. Report RZ1186, (Nov. 2, 1982) IBM Zurich Research Lab, 8803 Rueschlikon, Switzerland.

[46] Y. H. Viemont and G. J. Gardarin, A Distributed Concurrency Control Algorithm Based on Transaction Commit Ordering, Proc. 12th IEEE International Symposium on Fault-Tolerant Computing (1982) 81-88.

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

ANNA HAC

Anna Hać received her M.S. and Ph.D. degrees from the Technical University of Warsaw in 1977 and 1982. From 1983 to 1984 she was a postdoctoral fellow at the University of California at Berkeley. In 1984 she joined the faculty at the Johns Hopkins University where she is an Assistant Professor of Electrical Engineering and Computer Science.
Dr. Hać's research is in system and workload modeling, performance evaluation, reliability, modeling distributed file systems, modeling process synchronization mechanisms for distributed systems, and system simulation. Dr. Hać is a member of the Association for Computing Machinery and the IEEE Computer Society.

Address: Department of Electrical Engineering and Computer Science; The Johns Hopkins University; Baltimore, Maryland 21218.