M.I.T. Laboratory for Computer Science
Computer Systems Research Division

March 30, 1977
Request for Comments No. 141

MEASUREMENTS OF SHARING IN MULTICS

by Warren A. Montgomery
Room 510
M.I.T. Laboratory for Computer Science (formerly Project MAC)
Massachusetts Institute of Technology
545 Technology Square
Cambridge, MA 02139

This is a draft of a paper prepared for submission to the Sixth ACM Symposium on Operating Systems Principles, to be held at Purdue University, on November 16 - 18, 1977.

This research was sponsored in part by the Advanced Research Projects Agency (ARPA) of the Department of Defense under ARPA Order No. 2095 which was monitored by the Office of Naval Research under contract No. N00014-75-C-0661.

Measurements of Sharing in Multics.

ABSTRACT

There are many good arguments for implementing information systems as distributed systems. These arguments depend on the extent to which interactions between machines in the distributed implementation can be minimized. Sharing among users of a computer utility is a type of interaction that may be difficult to provide in a distributed system. This paper defines a number of parameters that can be used to characterize such sharing. This paper reports measurements that were made on the M.I.T. Multics system in order to obtain estimates of the values of these parameters for that system. These estimates are upper bounds on the amount of sharing and show that although Multics was designed to provide active sharing among its users, very little sharing actually takes place. Most of the sharing that does take place is sharing of system programs, such as the compilers and editors.

Key Words and Phrases: distributed systems, Multics, program behavior, virtual memory, measurements, sharing.

CR Categories: 4.35, 4.6

1. Motivation

One of the driving forces in the development of large, central computer systems was the high cost of computer hardware. Current trends in electronic technology indicate, however, that hardware costs are no longer the dominant factor in the design of an information system.

There are many reasons to implement a large information system as a distributed system, with several independent, communicating machines. These reasons include:

I. Increased reliability. Each machine in a distributed system would be smaller and simpler than the single machine in an equivalent central system. Thus the individual machines would be less likely to fail, and a failure in one machine need not disrupt all system operations.

II. Ability to grow. Adding a new machine to a distributed system can increase the computing power available without affecting the existing system.

III. Distributed authority. The availability and protection of information managed by each machine in a distributed system can be independently controlled.

IV. Better response. The parallelism available in a distributed system allows it to respond more rapidly than a multiplexed centralized system. Communication delays also can be reduced by placing the information and computing power physically near where they are needed.

All of the arguments for choosing a distributed organization for an information system are dependent on the extent to which that system can be separated into independent components. Many of the advantages of a

distributed system disappear if a significant amount of essential information is shared by components that are implemented on separate machines.

Sharing of information by processes executing on separate machines requires communication in order to keep the shared information consistent. Some protocols for maintaining consistency have been developed recently, but the problem is still poorly understood [1], [2], [3].

Current centralized systems can be divided into components, such as the files and processes belonging to individual users of a computer utility. However, the extent to which the components are independent is uncertain, because information is shared between components. Thus, before considering a distributed implementation for an information system, it is important to know the nature of sharing in that system. This paper reports an attempt to measure sharing in a large, centralized computer system, specifically, the Multics system.

2. Definitions

Before attempting to measure sharing, it is necessary to have a definition of what should be measured. A unit of sharing is called an object, which refers to a logical unit of information. The definition of sharing that follows is an attempt to capture the aspects of sharing that may be difficult to provide in a distributed system.

> An object is said to be shared in a time window t if it is referenced by at least 2 different processes within the time t.

It is useful to distinguish two kinds of references to an object: writes, which modify the contents of an object, and reads, which do not. This distinction allows us to define two specialized kinds of sharing:

write-sharing, in which an object is shared and is also written by at least one process, and read-sharing, in which all references to the object are reads. Write-sharing is inherently the more difficult of the two to provide in a distributed system, because it requires communication among the processes that are referencing the shared object, while read-sharing does not.

The time window in the definition of sharing is a very important parameter. This time window is an indication of the communication bandwidth needed to provide sharing. A much higher bandwidth is needed to provide sharing in a small time window than is needed to provide sharing in a large time window. The shared memory of a centralized system provides a very high bandwidth, while the bandwidth available in a distributed system is generally lower. Thus we are most interested in examining sharing in small time windows, which require higher bandwidth than is available in a distributed system.

Finally, it is necessary to measure the sharing of different types of objects separately, because different types of objects are referenced in different ways, some of which may be more difficult to provide in a distributed system. The types of objects considered are segments, directories, and pages.

3. Goals

Having chosen a definition of sharing, we can now ask what parameters best characterize the amount of sharing that takes place in a computer system. The parameters described below appear to be most useful for this purpose.

One parameter that is definitely of interest is the percentage of all objects of each type supported by the system that are shared in a given time window. The magnitude of this parameter increases as the time window is

increased. An infinitely long time window gives the percentage of all objects of a given type that are shared at some point in their lifetime. This parameter can be used to decide whether or not provisions for sharing should be made at the time that an object of that type is created. Smaller time windows give percentages of objects involved in more active sharing.

A second parameter of interest is the percentage of all object references that reference shared objects. This parameter provides an indication of the number of references that may require special treatment in a distributed system.

Another parameter that we wish to measure is the average number of processes that share a typical object. If a shared object is implemented by a multiple-copy strategy, then this parameter can be used to estimate the number of copies required and the communication that may be needed to keep the copies consistent.

As noted above, we desired to measure the extent of sharing in a large, centralized system. The Multics system is a good selection for these measurements for several reasons:

The Multics system was designed to provide extensive facilities for controlled sharing, more so than other commercially available computer systems. Thus an examination of Multics should provide an upper bound on the amount of sharing in current systems.

Multics also provides a good programming environment for the development of programs to obtain and analyze data on sharing.

The reported experiments were performed on the M.I.T. Multics system. However, the measurement tools developed could easily be used on other Multics sites.

4. Measurement Techniques

Knowledge of the parameters described above would allow us to estimate the storage capacity and communication bandwidth needed to provide sharing in a distributed system. Unfortunately, the Multics system does not provide the information necessary to measure these parameters directly: they must be indirectly measured or estimated. This section outlines the feasible measurements, and how these measurements relate to the parameters described above. It also provides a brief description of the virtual memory management algorithms used in the Multics system, which yield some data on the use of objects by processes. More detailed descriptions of these algorithms and the data bases that they maintain appear in [4] and [5].

The measurements of sharing were made by observing "snapshots" of the M.I.T. Multics System. A snapshot is a copy of the state of the system at some instant in time. Two methods were used to obtain such snapshots: copying the data bases of interest from a running system, and extracting them from the system dumps generated at the time of a crash. The data obtained from these two sources appear to be consistent.

The results presented in this report are averages of data obtained from nine snapshots taken between January and March of 1977. For most of the results reported, the data in individual snapshots varied less than 5% from the reported average. The results with larger variability are noted as they are presented in the next section. At the time of each of these snapshots, the system was loaded with approximately 50 users, many of whom were involved in program development, and thus were editing, compiling, or debugging programs. Another large class of users was involved in document preparation.

The Multics supervisor provides three primitive types of objects:

segments, which are logical information containers holding up to 255K 36-bit words, directories, which form the hierarchy used to organize segments, and pages, which are information containers holding 1024 36-bit words. Each page is part of some segment or directory and contains part of the information content of that segment or directory. The Multics virtual memory mechanism maintains detailed information about the use of segments and directories by processes for approximately 1000 segments and directories. This information is contained in a software-implemented cache, known as the Active Segment Table (AST), which is managed by an LRU algorithm. The segments and directories that are described in this cache are referred to as active.

For each active segment, the AST contains a list of the processes that have referenced that segment since it last became active. If two different processes appear on the list for a particular segment, then that segment has been shared, by the definition of sharing given above, in the length of time that the segment has been active. Unfortunately, the length of time that a particular segment has been active cannot be determined directly from the available information and must be estimated. This time varies from a minimum of about 1 minute, to several hours. The mean time is about 6 minutes.

Similar information is available for the active directories. Processes do not in general reference directories explicitly, but a reference to a segment implicitly references all directories above that segment in the file system hierarchy. A directory is guaranteed to remain active as long as there is some segment below that directory in the hierarchy that is active. For this reason, directories remain active longer than do segments. The average length of time that a directory was active in the system being measured was about 25 minutes.

The amount of write-sharing of segments can be estimated by examining the access privileges that processes have to shared segments. This information is available in the AST. This only gives an upper bound on the amount of write-sharing, as a process that has write access to a segment has not necessarily written that segment.

There is less information available about the use of pages by processes. An estimate of the sharing of pages can be made by considering a page to be shared if the segment or directory of which that page is part is shared, that is, all pages of shared segments and directories are considered to be shared. This estimate is again an upper bound, as most segments and directories have more than one page, and all processes that reference a segment or directory do not reference all of its pages.

We can easily estimate the magnitude of two of the sharing parameters described in the previous section from the data described above. The data give the percentage of objects that are shared, and the average number of processes that share an object. A good estimate of the percentage of object references that reference shared objects cannot, however, be obtained from the available data.

5. Results

This section presents the results of the sharing measurements discussed above. As previously noted, the data reported are averages for nine snapshots, each of which yields results very close to the reported averages. These results are analyzed to determine the causes of any unexpected results, and to determine the implications of all results for the implementation of a distributed Multics-like system.

5.1. The Sharing of Segments and Directories

Table 1 shows the total numbers and percentages of the active segments and directories that were shared in all nine snapshots. The table shows that about 59% of the active directories were shared, while only 12% of the segments were shared.

Table 1

Sharing of Active Segments and Directories.

	Unshared	Shared
Segments	5026 (88%)	705 (12%)
Directories	1034 (41%)	1525 (59%)

One reason for the larger proportion of shared directories is that each reference to a segment implicitly references all of the directories above that segment in the file system hierarchy. Thus a directory is shared if some segment below it is shared, or if two different segments below it are referenced by different processes.

Another reason for the high proportion of shared directories is the presence of system processes, which perform such functions as making backup copies of recently modified segments, and creating processes for users. These system processes reference many directories, but do so very infrequently. A later section of this report discusses the effects of such system processes on our measurements.

5.2. Read-Sharing and Write-Sharing of Segments

Table 2 gives the numbers and percentages of active segments that were read-shared or write-shared.

Table 2

Read-Sharing and Write-Sharing of Segments

Read-Shared	613	(10.6%)
Write-Shared	92	(1.6%)

These figures suggest that very little write-sharing of segments takes place. Most of the write-shared segments were in fact being used to implement system supported higher level data structures, such as mailboxes. These segments are referenced only by system supplied programs. These programs synchronize their writes with a simple protocol that could be implemented easily in a distributed system.

5.3. The Effect of System Processes and Segments

The measurements of sharing presented above include all of the processes, segments, and directories in the Multics system. We would like to be able to examine sharing of objects by user controlled processes separately from sharing due to the system processes mentioned above. A significant amount of the sharing due to system processes could probably be avoided in a distributed system by duplicating the system processes at each site. Thus the separation of the sharing that is solely due to the interaction of user-controlled processes is important.

It is also interesting to examine the sharing of user-created objects separate from that of system objects such as system programs or subroutine

libraries. Many system objects are very rarely modified, and therefore could easily be duplicated at each site in a distributed system; or they are directly related to the functioning of the machine, and thus would not be shared across site boundaries in a distributed system.

Table 3 gives the results of performing some of the measurements reported in Tables 1 and 2 on the same system (a) with the effect of system processes removed, and (b) with both the system processes and system segments and directories removed. The table gives the total numbers and percentages of read-shared and write-shared active segments and of active directories in the nine snapshots. The table gives three sets of figures, one for each of the restricted cases mentioned above, and a summary of the results reported in Tables 1 and 2 for comparison.

Table 3

The Effect of System Processes and System Objects on Sharing Measurements

	All Processes, All Objects	User Processes, All Objects	User Processes, User Objects
Read-Sharing of Segments	613 (10.6%)	585 (11.6%)	66 (2.0%)
Write-Sharing of Segments	92 (1.6%)	33 (0.6%)	6 (0.2%)
Sharing of Directories	1525 (59%)	693 (31%)	564 (24%)

These figures show that the system processes are responsible for a substantial proportion of the sharing of directories, and most of the write sharing of segments. One system process known as the Initializer references a large number of directories that are referenced by only one other process. Thus removing the effects of the Initializer process greatly decreases the

apparent directory sharing.

The figures above also show that most of the read-shared segments are system segments. Each process references a large number of system programs. The segments that contain these programs account for most of the read-sharing.

5.4. The Average Number of Processes that Share an Object

The figures presented so far suggest that sharing of segments may not be very important in Multics. The percentages of shared segments seen here suggest that relatively rarely is a particular segment shared. It is possible, however, that the shared segments are much more heavily used than the unshared segments and are therefore very important. One way to estimate such an effect is to measure the number of processes that share each active segment and directory, and to compute the average. Table 4a gives this average for segments and directories for the three cases indicated in Table 3.

Table 4a

The Average Number of Processes that Share an Average Segment or Directory.

	Segments	Directories
All Processes, All Objects	2.6	4.3
User Processes, All Objects	2.6	3.6
User Processes, User Objects	1.1	2.3

The figures in Table 4a suggest that the shared segments are shared by many processes. Although few segments and directories are shared, the average number of processes that share a segment or directory is large. This fact is even more apparent if we compute the average number of processes that share a

shared segment or directory. (The averages in Table 4a are for all segments and directories, shared or unshared.) These figures appear in Table 4b.

Table 4b

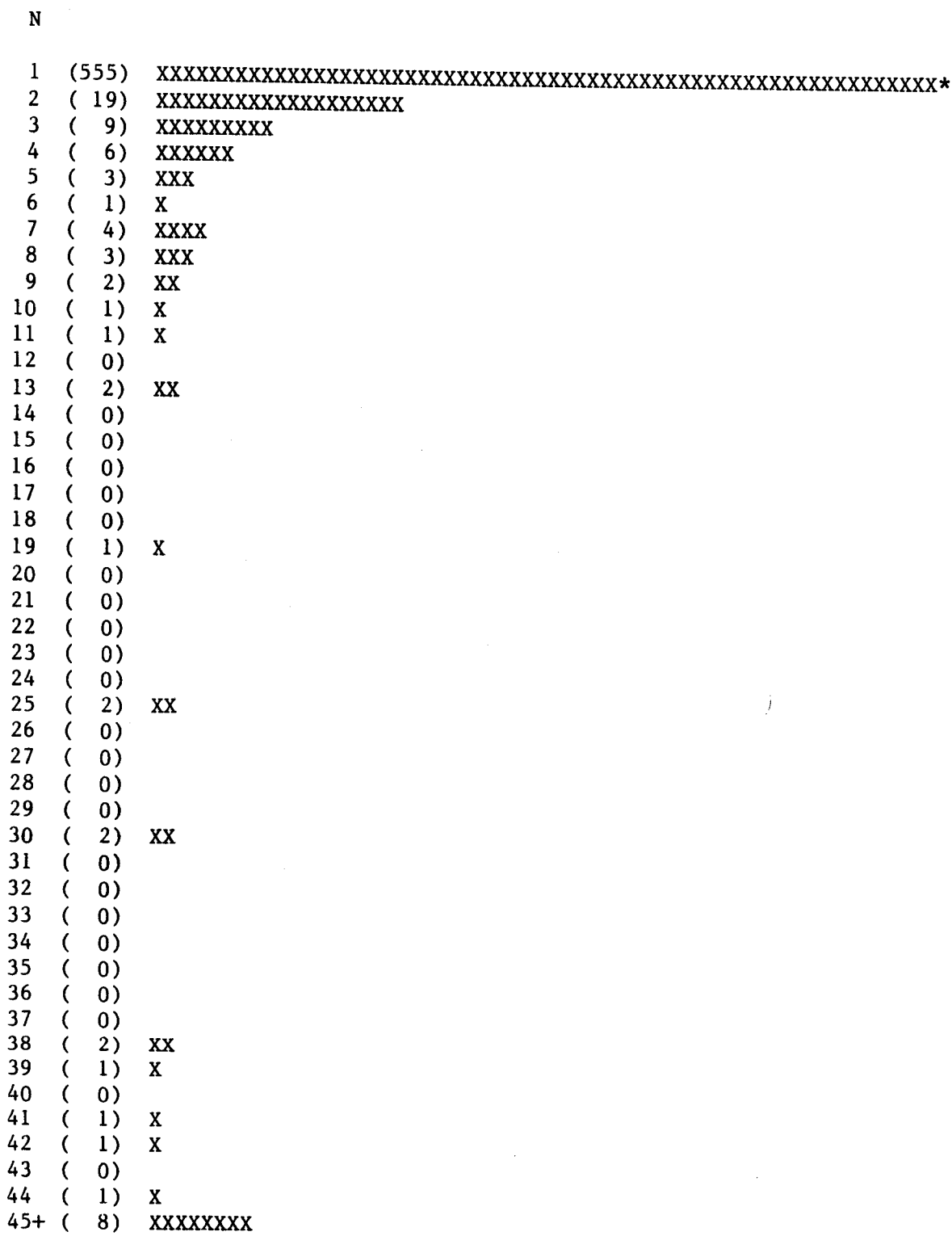
The Average Number of Processes that Share a Shared Segment or Directory.

	Segments	Directories
All Processes, All Objects	13.9	6.6
User Processes, All Objects	13.8	6.7
User Processes, User Objects	3.6	3.1

Figure 1 shows a histogram of the data from one snapshot used to derive the average for all processes sharing all segments. The figure shows the number of segments that were shared by exactly N processes as N varies from 1 to 45. Notice that most of the segments are unshared or shared by only 2 processes, while a few are shared by most of the processes on the system. These few are the segments that contain the most popular system programs.

Figure 1

The Number of Segments Shared by Exactly N Processes



5.5. Sharing of Pages

As previously noted, information on the sharing of pages by processes is not directly available, and must be estimated based on the use of segments and directories. The three-level memory hierarchy used by Multics allows us to distinguish three sets of pages for which sharing parameters can be estimated. Each level of the memory hierarchy is managed by a page replacement algorithm that closely approximates LRU, so as to keep the most recently referenced pages in the lowest levels of the hierarchy. In the snapshots analyzed in this report, all pages in main memory (core) had been referenced within the last 0.5 second, and all pages in the second level of the hierarchy, the paging device (PD), had been referenced in the last 4 minutes. Pages in the third level (disk) had not been referenced in the last 4 minutes. These time intervals should not be confused with the time window of sharing, which is the same for segments and directories as for their pages. These intervals indicate the time at which a page was last referenced by some process, but not the sharing of a page by two or more processes.

Table 5 reports upper bounds on the amount of read-sharing and write-sharing of pages referenced in the last 0.5 second, and of those referenced in the last four minutes. The table shows that 361 pages were referenced in the last 0.5 second and were read-shared, which represents 29% of all of the pages that were referenced in the last 0.5 second.

The data reported are averages of data obtained from nine snapshots. The data obtained from some of these snapshots varied considerably from these averages. (The percentage of pages referenced in the last 0.5 second that were read-shared, for example, varied between 15% and 57%.) This variability

can be explained by the fact that the references observed in any 0.5 second interval may not be typical of all references made in the system. This interval is small compared to the length of time that processes are blocked while waiting for interactions with users, and thus few processes actually make references in the 0.5 second interval from which the data is taken.

Table 5

Sharing of Pages

	Referenced in Last 0.5 Second	Referenced in Last 4 Minutes
Read-Shared	361 (29%)	2897 (22%)
Write-Shared	12 (1.3%)	113 (0.8%)

Notice that the pages more recently referenced are more likely to be shared. We expect that shared pages are more frequently referenced, as they are referenced by more than one process.

We can also compute the average number of processes that share a page at each level of the memory hierarchy. This is done by computing the sum for each memory level of the number of processes that share each page at that level (estimated, again, on the basis of what we know of the sharing of the corresponding directory or segment), and dividing that sum by the number of pages at that level to obtain the average. Table 6 presents the average computed for pages in each level of the memory hierarchy.

Table 6

The Average Number of Processes Sharing a Page.

	Pages of Segments	Pages of Directories
Core Pages	6.6	28.8
PD Pages	5.2	17.1
Disk Pages	2.3	5.7

Notice that the number of processes that share a page is highest in the fastest portion of the memory hierarchy. This is because the chance that a page will be referenced increases with the number of processes that share that page, and thus shared pages compete more effectively for space on the paging device and in core.

Note that the figures in this table are much higher than those for the average number of processes sharing segments or directories reported in Table 4a. This difference is due to the fact that shared segments and directories tend to have more pages in general than unshared segments and directories. Therefore, with the method used to estimate the sharing of pages, a higher proportion of pages than of segments or directories are counted as shared.

6. Conclusions

The results in the previous section show that very little sharing of user-created objects takes place among user-controlled processes on the M.I.T. Multics system. This fact suggests that it may be possible to implement a Multics-like system in a distributed environment where sharing of memory pages is not possible.

There is very little write-sharing of segments. Most of the write-shared segments are referenced only by system programs, which use simple protocols to synchronize their writes.

Although a higher proportion of directories than of segments are shared, it is unlikely that shared directories would cause as much trouble in a distributed system as would shared segments. The difficulty of implementing a shared object in a distributed system is related to the size of that object and the frequency of updates. For the system being measured, the average size of an active segment (8.6 pages) was much larger than the average size of a directory (3.9). More importantly, although implicit references to directories are quite frequent, updates to directories are very rare.

Another point to consider is that the supervisor implements updates to directories, and thus takes care of synchronizing those updates. A different synchronization mechanism would be needed in a distributed system, where several copies of a directory may exist on different machines. However, only a few system programs need be aware of how this synchronization is performed. The users and user-written programs do not need to know the details of the protocols used to synchronize updates.

Processes reference segments with machine instructions and not calls to the supervisor. The supervisor makes no attempt to synchronize updates to a segment. Instead, the processes that write-share a segment depend on the fact that they are sharing the same copy of the segment, and use hardware instructions that act on the segment to perform synchronization. A different synchronization strategy would be required in a distributed system, and all programs that use write-shared segments would have to be modified to adopt the new strategy. This problem does not arise in a more conventional file system

where all references to files are performed by calls to the supervisor, and not by machine instructions.

Although very little sharing of user-created objects takes place in the M.I.T. Multics system, the results reported show that many system segments are intensely shared. Copies of these frequently referenced system segments would have to be maintained at each site in a distributed system in order to achieve satisfactory performance. While the cost of storage is rapidly decreasing, the extra storage required to hold the copies may be a significant expense.

References

1. Alsberg, P.A., Belford, G.G., Day, J.D., Enrique, G. Multi-copy resiliency techniques. University of Illinois, Center for Advanced Computation Document No. 202 (May 1976).

2. Johnson, P.R., and Thomas, R.H. The maintenance of duplicate databases. Arpanet Network Working Group RFC No. 677 (January 1975).

3. Thomas, R.H. A solution to the update problem for multiple copy data bases which uses distributed control. Bolt Beranek and Newman, Inc. Technical Report No. 3340 (July 1976).

4. Bensoussan, A., Clingen, C.T., and Daley, R.C. The Multics virtual memory: concepts and design. Comm. ACM 15, 4 (May 1972), 308-318.

5. Organick, E.I. The Multics System: An Examination of its Structure. MIT Press, Cambridge, Mass, 1972.