NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

ISBN 0-315-59181-1

CICS LSR Buffer Simulator (CLBS)

George Bozikian

A Major Project

in

The Department

of

Computer Science

Presented in Partial Fulfilment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

August 1990

ABSTRACT

CICS LSR BUFFER SIMULATOR (CLBS)

G. Bozikian

Buffering is a technique used to reduce the number of I/O operations, but deciding on how many buffers to allocate is difficult. Adding buffers should decrease the number of I/O's, but each buffer uses up memory. CLBS has been designed to allow users of Local Shared Resources (LSR) in the Virtual Storage Access Method (VSAM) environment to simulate different-sized buffer pools, thus eliminating the uncertainty.

CLBS consists of two sets of programs. The first set intercept all LSR I/O's and record the address (RBA), the name of the file, and the LSR pool being used. The second set of programs use the data gathered above to find the number of physical I/O's that would have been performed had a different number of buffers been allocated.

CLBS was used to simulate changes to the number of buffers in different systems. The changes were then implemented and it was found that the predictions made by CLBS were accurate.

TABLE OF CONTENTS

INTRODUCTION

Advances in electronic components have led to very large increases in CPU speeds. However, Direct Access Storage Devices (DASD) contain mechanical components, therefore the increases in DASD access times have been small in comparison. Many software and operational solutions have been devised to reduce the time taken by the mechanical components, e.g., the scheduling of queued I/O requests [LOC80, ATW82] and the placement of data on DASD [IBM78]. The most successful method has proved to be the elimination of a portion of the I/O requests.

This elimination of I/O requests is accomplished by keeping the most recently used records in memory areas called buffers, such that a re-access to the same record can be satisfied without a physical I/O operation. The amount of time required to service an I/O request is a few microseconds in case of a hit (i.e., record found in a buffer), but is about 40 milliseconds in case of a miss (i.e., record not found in a buffer). Due to this large difference in service times, achieving a large number of hits will dramatically improve response times.

The decision on the number of buffers to be allocated is difficult. One is tempted to allocate many buffers so as to minimize the number of DASD accesses. However, buffers occupy virtual memory space, which, like I/O, is a resource. Overallocation would lead to more system overhead to perform the additional paging and virtual storage management. This additional overhead will lead to response time degradation [DAR01, BER87] and degraded people productivity [DOH01].

The statistics available to the person making the decision of how many buffers to allocate, is usually limited to the number of I/O requests that resulted in physical I/O operations, and the number of requests that were obtained from the buffers without physical I/O's. System programmers tend to avoid changing buffer allocations because:

- The statistics tend to vary from day to day depending on the sequence of transactions issued by the users during a day.
- The buffers are allocated at system startup time and cannot be changed dynamically, therefore new buffer allocation strategies take a long time to implement.
- It is risky to implement changes on production systems because a change in the size and number of buffer allocations can just as easily degrade performance as improve it, e.g., through increased paging.

The simulator presented in this paper allows system programmers to test different buffer allocation strategies and to choose the optimum one. The simulator consists of two phases: the first phase records information about all I/O requests, while the second phase consists of a program that mimics the Least Recently Used (LRU) replacement algorithm to evaluate whether an I/O request would have been fulfilled from memory or from DASD.

The CLBS simulator models an IBM Customer Information Control System (CICS), however the model for this simulator can be used to simulate any memory buffering system that uses the LRU replacement algorithm.

1. Definition of DASD Concepts.

This chapter describes the components of a DASD I/O operation, then proceeds to outline techniques for improving DASD responsiveness.

1.1 Description of DASD I/Os.

Figure 1.1 shows the components involved in an I/O to DASD. The processor is connected to the I/O subsystem via channels, which in turn connect to DASD controllers, and finally to the DASD units where the data reside. Each DASD unit consists of a number of rotating platters, each having their own read/write heads, (Figure 1.2).

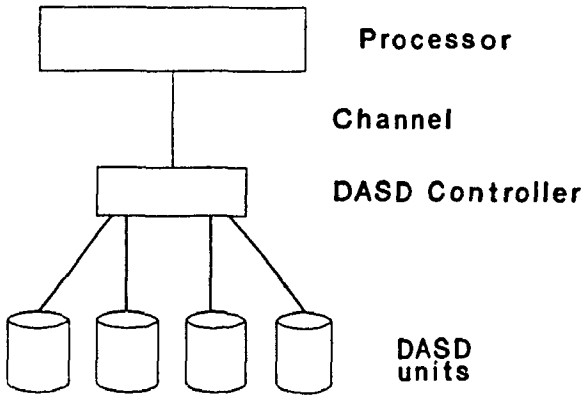


Figure 1.1 Components Involved in DASD IO

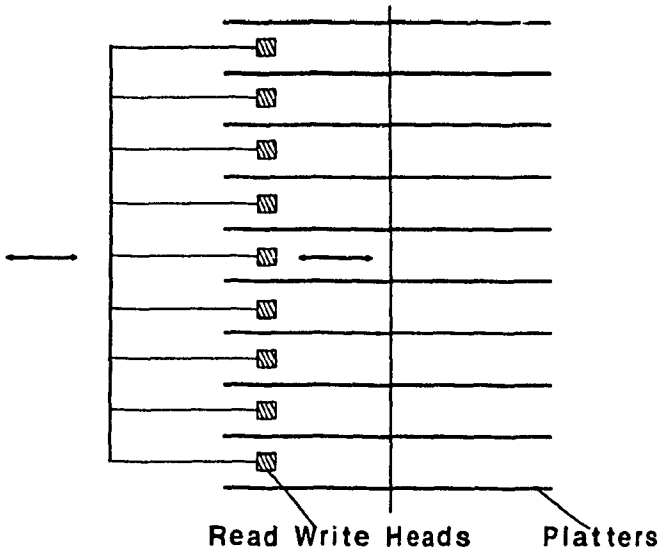


Figure 1.2: Side View of a DASD Unit.

The 4 phases of a DASD I/O operation are described here:

1) the channel is used to send a request from the CPU to the DASD controller.
2) the read/write head of the DASD is moved to a specific depth (cylinder) on the device. The time required to perform this activity is called seek time.
3) the device waits until the platter rotates such that the data to be read or written is under the head (Figure 1.3). The time spent waiting for the data to rotate into position is called rotational delay.
4) finally, the channel is used to transfer the data from the DASD to the CPU.

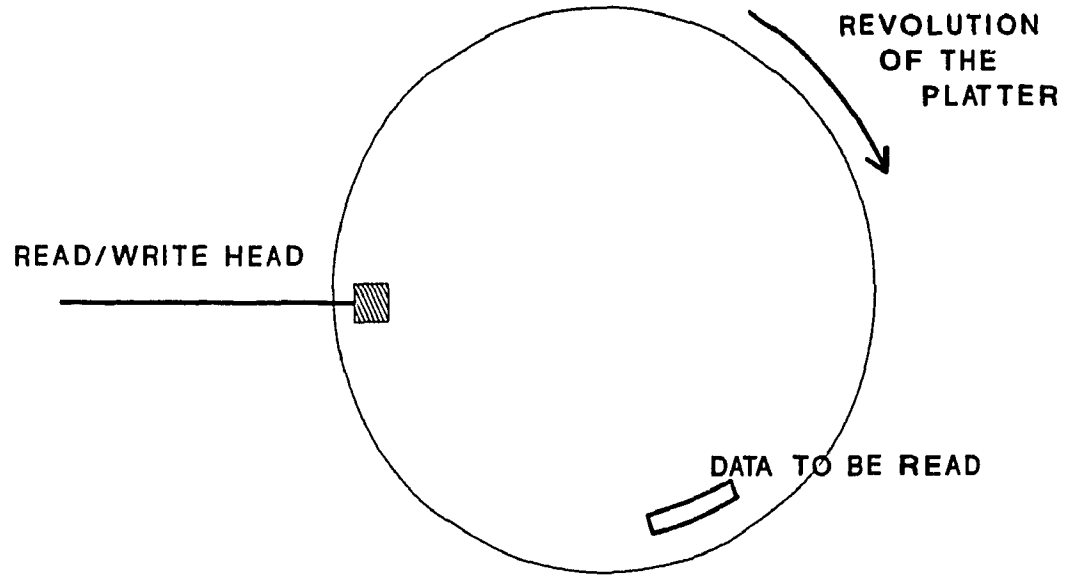


Figure 1.3: Top View of a DASD Platter.

The activities described in points 2 & 3 above are mechanical, therefore they are the activities which take the longest to complete.

1.2 Improving DASD I/O Responsiveness.

An interactive user's productivity is impacted directly by the speed with which he receives the response. Looking at a breakdown of response time we find three major categories:

- Network time,
- DASD I/O time,
- Computing time.

Considering that business applications spend at least half the time waiting for data to be accessed from, or written to, DASD, it is not surprising that hundreds, if not thousands, of articles and books are written every year on ways to speed up DASD accesses. Listed here are some of the more important techniques, [SMI81a] provides a detailed history on this subject for IBM mainframe systems:

- Hardware changes to speed up the channel, the movement of the head, and the rotation of the platter.
- Data placement so that the most frequently accessed data are placed together [IBM78].
- Scheduling of I/Os so that the data that are closest to the head are accessed first [LOC80,ATW82].
- Predicting data that might be required and pre-fetching them into memory.
- Introduction of caching in controllers, thus eliminating the seek and rotational delays [SMI81b].

1.3 DASD I/O Avoidance.

Compared to the large improvements in CPU hardware speeds, there has been very little improvement in DASD hardware performance. This has led designers to focus on using CPU hardware (processor and memory) to reduce the need

for DASD accesses. These designers used memory to store images of the most recently accessed records (called Data Buffering). Every time an I/O request is issued, a search is performed through memory first and an I/O performed only when the required record is not found in memory. These re-accesses from memory are a result of two factors as described in [SMI81b],

- recently used information is retained, providing locality by time.
- the buffer contains blocks of information rather than just the words or bytes that were accessed, information that is near the referenced data is also available in the buffer.

Data buffering has been very successful in improving access to data, and all large data base systems have adopted it so as to provide acceptable response times to interactive users.

However, data buffering does not come free. Memory must be used to store the images, therefore the amount of data buffering must be carefully chosen so as to minimize this cost. This paper provides a model of a simulator which can be used to predict the behavior of a system with varying amounts of memory dedicated to data buffering.

2. DESCRIPTION OF ENVIRONMENT

Though the buffer simulation methodology presented in this paper is applicable to any system using data buffering, the implementation was conducted on an IBM system shown in Figure 2.1. This environment was chosen because of its availability to the researcher, and due to the preponderance of such systems in use.

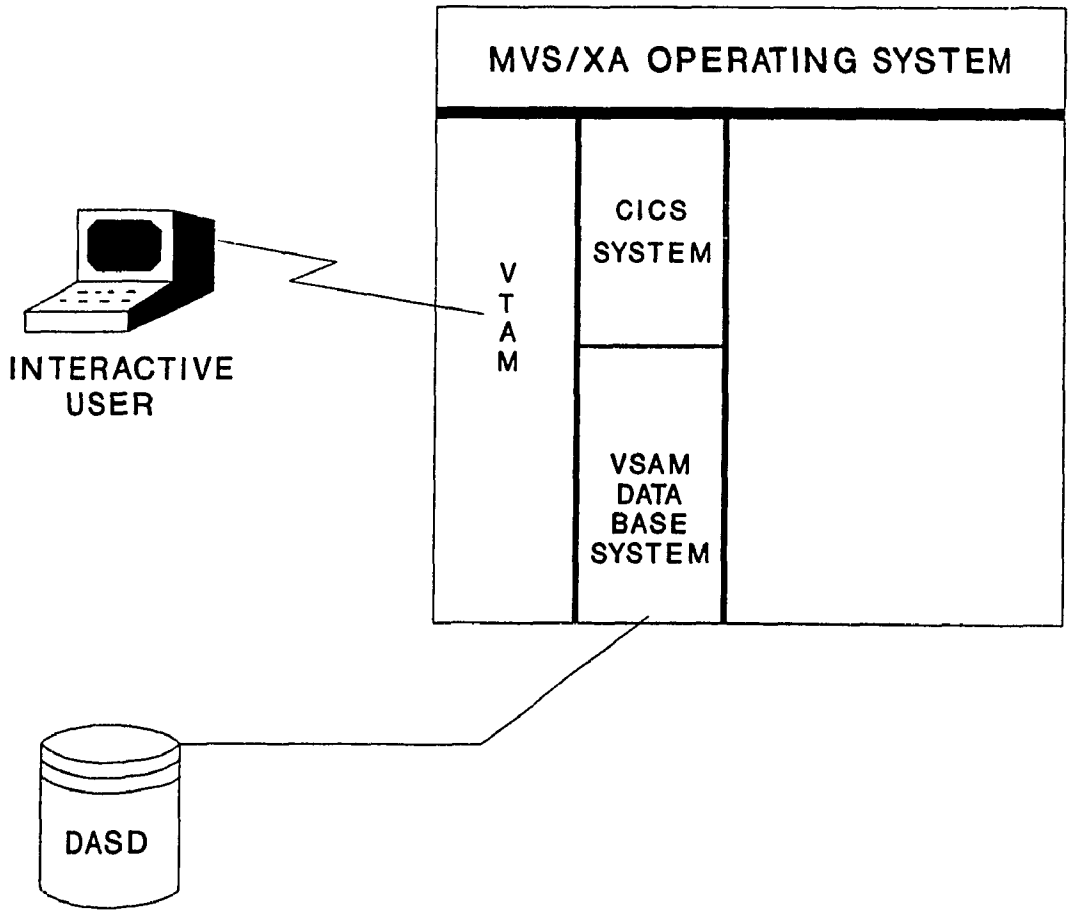


Figure 2.1: Operating System Environment

The figure shows that the environment consists of,

- Operating system: IBM Multiple Virtual System/Extended Architecture (MVS/XA system),
- Networking: IBM Virtual Telecommunications Access Method (VTAM),
- Support for terminal based applications: IBM Customer Information Control System (CICS),
- Data Base Access method: IBM Virtual Storage Access Method (VSAM).

This chapter describes the VSAM access method and its data buffering facility called Local Shared Resources (LSR). These descriptions will then be used to define a model of CLBS.

2.1 Virtual Storage Access Method (VSAM)

VSAM is an access method which performs DASD I/O operations based on requests it receives from user programs (in this case from CICS user programs). The data buffering function in VSAM is called LSR, and it is this function which is simulated by the programs presented in this paper. Figure 2.2 shows the organization of a sample VSAM file. The number of levels of indices is variable,

and in some files does not exist, in which case the file can only be accessed sequentially.

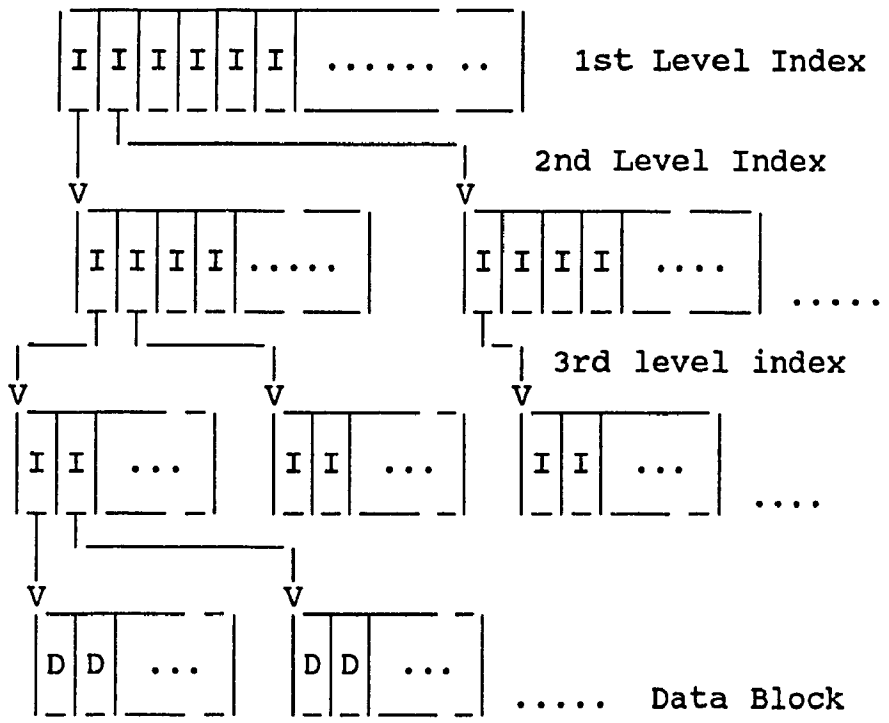


Figure 2.2: The Structure of a Sample VSAM File where I is an Index (Pointer) and D is a Data Record.

When an I/O request is received by VSAM it will first issue the appropriate I/O commands to bring in the highest index level (assuming that the file is not sequential). Once the highest index level is in memory, VSAM will use the key ranges in the index to find the index at the next level (if any) and so on until it finds a pointer to the block of data

containing the required record. At this point the block of data will be brought into memory and the requested record will be made available to the application.

2.2 Local Shared Resources

The LSR option of VSAM provides the data buffering functions described earlier. Therefore, whenever an I/O operation is performed VSAM first searches the LSR buffers to see if the required data (or index) record is already in memory.

When LSR is used, buffers will be shared by a number of files, rather than having a different set of buffers for each file. Through this sharing, memory requirements will be reduced, and the more active files will end up with a higher proportion of the buffers.

Figure 2.3 shows an example of the type of data which are supplied by CICS & VSAM, showing the utilization of the different LSR buffer pools. The data are used to make decisions about LSR buffer allocations, with the intent being to either reduce the number of I/O operations to DASD or to reduce the amount of memory allocated to the buffers.

Buffer Size	# of Buff	# of Buff Hits	# of Buff Miss
512	28	2,634	17
1,024	15	16,082	34
2,048	18	15,041	3,645
4,096	18	1,900	14
8,192	14	1,043	133
TOTALS		36,700	3,843

Figure 2.3: Sample of LSR Pool information

The data shown above (as well as all other data in this project) comes from CICS systems running at Canadian National. As can be seen in Figure 2.3, the system has been defined with five different pools of buffers. There are actually seven but two handled no I/O requests during the test periods.

The system programmer uses these data to decide how many of each type of buffer to allocate. The success or failure of changes can be measured by the 'Hit' ratio:

$$\text{Hit Ratio (HR)} = \frac{\text{\# of Buffer Hits}}{\text{\# of buffer Hits + \# of Buffer Misses}}$$

In the case above (Figure 2.3):

$$HR = \frac{3,843}{36,700 + 3,843} = 91\%$$

It is the system programmer's responsibility to allocate a number of pools of buffers. Each pool consists of a number of equal-sized buffers; in Figure 2.3, for example, 28 x 512 byte buffers, 15 x 1,024 byte buffers, etc., have been allocated. When a block of data is required to be brought into memory, VSAM will use a buffer from the buffer pool with the appropriate size; in Figure 2.3, a 512 byte block would use a buffer from the set of 28 x 512 byte buffers.

2.3 Model of CLBS Implementation

The logic used by CLBS to simulate LSR buffering is presented in this section. For each simulation, the user provides the buffer pool (size), number of buffers, and an input file which contains the sequence of I/O requests issued during a normal execution of the system.

Based on the user's input, CLBS builds an array equivalent in size to the number of buffers to be simulated, and then proceeds to read the input file containing the sequence of I/O requests. For each I/O request, CLBS first checks to see if the requested record is already in a buffer (i.e., array), if not, the new record replaces the record which has been unused for the longest period (LRU algorithm). However, if the record is already in a buffer, the buffer is moved to the top of the chain. Finally, in cases where the I/O request is for a file which contains multiple index levels, CLBS will repeat the insertion of an entry into the buffers for each of the index levels, up to a maximum of four levels.

3. TYPES OF SIMULATIONS

There are basically three techniques available to provide predictions of system performance [DEE84]. These techniques allow for the testing of theories before implementing the changes in a production environment. The three techniques are: Benchmarks, Simulations, and Analytical Models. These three techniques are evaluated in this section.

3.1 Benchmarking

Benchmarking consists of testing theories in a standalone mode, where user requests are generated (usually automatically), and performance measurements taken and compared to measurements taken before the changes were made. The steps which would be required to carry out a benchmark are outlines here:

- The state of the system and its data (permanent storage) are saved, this state will be referred to as STATE0.
- The system is run normally, and a record kept of the user requests and the attained performance.
- The state of the system is saved again, this state will be referred to as STATE1.

- STATE0 is restored and the change to be tested is applied.
- The user requests recorded earlier are now reissued, and a record kept of the attained performance.
- STATE1 is restored, and normal functioning resumes.
- The performance measures taken before and after the implementation of the change are compared and a decision made on whether or not to implement the change in production.

Of the three techniques being discussed here, a "Good" benchmark provides the most accurate predictions, however it is by far the most difficult of the three to carry out. As a result of the large investment required in time and hardware resources, benchmarking is rarely used in predicting performance.

3.2 Simulations

In simulations, the functions of a system are programmed, such that the program closely mimics the system being measured [DEE84]. The input to a simulation [LOC80] can either be a trace of events recorded during the normal functioning of the system, or randomly generated events.

Simulations are less costly than benchmarks, however the development time of simulators is still very costly when the system to be simulated is complicated. As far as the accuracy is concerned, a simulator is less accurate than a "Good" benchmark.

3.3 Analytical Models

Analytical modelling is based on queuing theory [KLE76] and was originally developed in the early 1900's to help telephone companies in predicting the effects of installing trunks between different cities. Though it is now heavily used to model computer systems, it is recognized that analytical models cannot properly reflect variations found in real time systems, e.g., morning versus afternoon activity. This deficiency makes Analytical Modelling the least accurate of the techniques, however it is the only feasible option available for complex systems because benchmarks and simulations are too costly to develop.

3.4 Prediction Technique Chosen

The discussion in this chapter has highlighted that Analytical Modelling is the only feasible technique for developing predictors for complex systems. However, the case at hand is a very simple system (LRU replacement algorithm), and therefore easy to develop a simulator for. Using simulation also provides additional accuracy [SAL86] not available through analytical modelling.

Next we needed to make a choice between the input formats, i.e., randomly generated events versus trace of events. Randomly generated events should only be used when an event trace cannot be easily obtained because trace driven simulators are more accurate than simulators driven by randomly generated events [LOC80]. In the case at hand, exit points are provided by CICS, which can be easily coded to provide a trace of most of the events of interest. Event traces were therefore chosen as the input format.

4. PHASES OF CLBS

There are two phases in CLBS, the first phase consists of the setting of traps within the CICS system to be simulated, so that data are gathered for each I/O request using the LSR buffers, while the second phase consists of running a simulation using the data gathered in phase 1. Since the data are gathered from interactive CICS systems, the data gathering is not allowed to use any locking (semaphores), because these locks would otherwise interfere with the operation of the CICS system being monitored. Since no locks are used, it cannot be guaranteed that all I/O requests are trapped.

The simulation phase of CLBS consists of using the data gathered in phase 1 to predict expected hit ratios, assuming a certain (user defined) number of buffers. Using the simulator, users can easily simulate different buffer allocation scenarios, allowing them to choose the most efficient scenario.

4.1 Data Collection Phase

IBM supplies exits at certain points in CICS so that users can add functions not supplied by IBM. Two such exits are used by CLBS to record all I/O requests to files in the LSR pool. The exits are:

1) XFCINC: Called whenever a read I/O completes.
2) XFCOUT: Called whenever a write I/O is initiated.

These two exits are used because they are given control immediately after CICS rearranges the buffers so that the buffer involved in the I/O is at the top of the chain of buffers. Every time one of the exits is given control, it accesses the buffer at the top of the chain and extracts the following information (see Appendix A for the control block structure used to collect this data):

1) FILENAME : Name of the file for which an I/O request was issued.
2) RBA (Relative Block Address): Identifies the block within the file for which the I/O request was issued.
3) SZOFBUF : Size of the buffer which is used.
4) NUMOFBUF : Number of buffers in this buffer set.
5) NUMOFRDS : Number of physical reads (i.e., buffer misses) which have been performed.

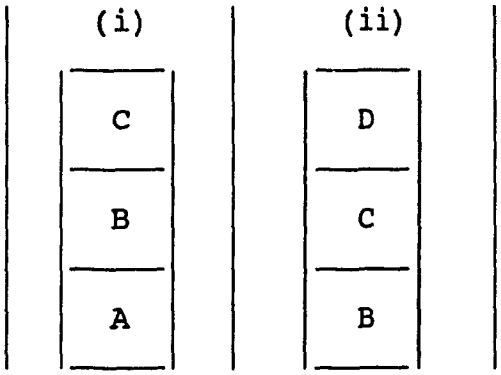
The exits, called POST, gather the data specified earlier and place them in a memory location. Whenever the area becomes full, the POST exits will switch to an alternate memory area, and will alert a program called WRTR, which will perform the actual writing to disk.

4.2 Simulator Program

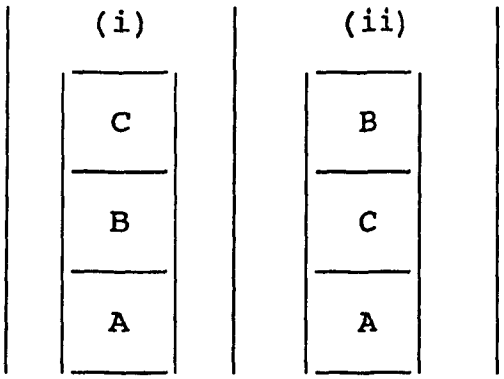
The simulation portion of CLBS is written in SAS (Statistical Analysis System) and basically keeps a number of buckets, equal to the number of buffers being simulated. Each bucket contains two pieces of information: the name of the file to which the record belongs and the block number within the file. These two pieces of information are enough to check whether or not the data are already in a bucket (buffer). The simulator handles the buckets in one of two ways, depending on whether the data were already in the buffers or not.

1) For a record not found in the buckets, the records are pushed down such that the oldest record falls off and is no longer available; the new record is placed at the top of the bucket. For example, at a certain point in time, the contents of the buffers are as shown below (i). A

request is made for record D, (ii) shows the buffer contents when the operation is completed.



2) For a record found in a bucket, the record is placed at the top of the chain of buckets. Consider the same buffer contents as above except that the request is for record B which is already in a buffer, (ii) shows the buffer contents when the operation is completed.



Since the simulator is driven by the events gathered by the exits, there is no need to use probabilities or random event generators.

5. Developing the Simulator

This chapter summarizes simulations performed with an early version of CLBS, and shows ways in which CLBS can be used. A more detailed account of these tests is provided in [BOZ88]. These tests were carried out to ensure the viability of the conceptual model of LSR functioning on which CLBS was based.

5.1 Initial validation

The simulator was run in a test CICS region (TESTEM) which provides Electronic Mail functions. Figure 5.1 shows the results of the actual runs as extracted from CICS, while Figure 5.2 shows a run of the simulator with the same buffer setup. Had the simulation been perfect, the values in Figures 5.1 and 5.2 would have been identical.

Hit Ratios (HR) were used to gauge the difference between the actual data extracted and the simulation runs (Figure 5.3). As can be seen the simulator was found to be highly accurate.

Buffers assigned during the runs							
Buffer sizes	512	1024	2048	4096	8192	TOT	
Number of buffers	28	15	18	18	14		
Number of bytes used for buffer	14K	15K	36K	72K	112K	249K	
Number of I/O requests	0	14864	18136	1472	1040	35K	JUNE 29
# of buffer misses (Actual I/O)	0	18	3624	3	118	3763	
Number of bytes read from DASD	0	18K	7248K	12K	944K	8.03M	
Hit ratio	N/A	100%	80%	100%	89%	89%	
Number of I/O requests	0	81755	90629	1704	1226	171K	JULY 2
# of buffer misses (Actual I/O)	0	59	19132	4	94	19289	
Number of bytes read from DASD	0	59K	38264K	16K	752K	38.17M	
Hit ratio	N/A	100%	79%	100%	92%	89%	
Number of I/O requests	38	18274	21784	2384	1564	43K	JULY 8
# of buffer misses (Actual I/O)	1	106	4372	18	183	4680	
Number of bytes read from DASD	.5K	106K	8744K	72K	1464K	10.14M	
Hit ratio	97%	99%	80%	99%	88%	89%	

Figure 5.1: Data extracted from CICS on three different runs.

Buffers assigned during the runs							
Buffer sizes	512	1024	2048	4096	8192	TOT	
Number of buffers	28	15	18	18	14		
Number of bytes used for buffer	14K	15K	36K	72K	112K	249K	
Number of I/O requests	0	14864	18136	1472	1040	35K	JUNE 29
# of buffer misses (Actual I/O)	0	21	3382	21	106	3530	
Number of bytes read from DASD	0	21K	6764K	84K	848K	7.54M	
Hit ratio	N/A	100%	81%	99%	90%	90%	
Number of I/O requests	0	81755	90629	1704	1226	171K	JULY 2
# of buffer misses (Actual I/O)	0	13	18095	9	59	18176	
Number of bytes read from DASD	0	6.5K	36190K	36K	472K	35.85M	
Hit ratio	N/A	100%	80%	99%	95%	90%	
Number of I/O requests	38	18274	21784	2384	1564	43K	JULY 8
# of buffer misses (Actual I/O)	2	79	4064	57	186	4388	
Number of bytes read from DASD	1K	79K	8128K	228K	1488K	9.69M	
Hit ratio	95%	100%	81%	98%	88%	90%	

Figure 5.2: Simulation of the three runs listed in Figure 3.

	System	Hit Ratio of actual run	Hit Ratio of simulation
June 29	TESTEM	89%	90%
July 02	TESTEM	89%	90%
July 08	TESTEM	89%	90%

Figure 5.3: Accuracy of the initial simulator.

5.2 Example of Usage

The data presented in Figure 5.2 permit drawing the following conclusions about the number of buffers allocated to each buffer pool:

512 byte buffers: The steady state is not being reached, which indicates that the number of buffers can be reduced.

1K byte buffers: There are a lot of requests to this buffer, but the fact that the hit ratio is so high (100%) indicates that the number of buffers here can also be reduced.

2K byte buffers: This set of buffers handles the largest number of requests, and the hit ratio is the lowest of any of the sets (81%). Therefore, this buffer set is an excellent candidate for expansion.

4K byte buffers: A very high hit ratio is being achieved (99%). Therefore, the number of buffers could be reduced.

8K byte buffers: I/O's in this buffer set are costly because of the large blocksize; also, the hit ratio is on the borderline (90%). Therefore, this pool should remain unchanged.

Note that Steady state in CLBS is defined to be the point at which all the buffers of a specific buffer pool have been filled. In the case shown in Figure 2.3, steady state is reached when 28 different 512 byte records have been accessed.

In summary using the data supplied by CICS, one would likely assign buffers as follows:

512x20 , 1Kx11, 2Kx38, 4Kx10, 8Kx14.

instead of the original allocation of:

512x28 , 1Kx15, 2Kx18, 4Kx18, 8Kx14.

Buffers assigned during the runs							
Buffer sizes	512	1024	2048	4096	8192	TOT	
Number of buffers	20	11	38	10	14		
Number of bytes used for buffer	10K	11K	76K	40K	112K	249K	
Number of I/O requests	1091	15651	17067	1769	1040	36K	JUNE 29
# of buffer misses (Actual I/O)	10	31	3290	60	106	3497	
Number of bytes read from DASD	2K	31K	6580K	240K	848K	7.52M	
Hit ratio	99%	100%	81%	97%	90%	90%	
Number of I/O requests	0	86472	87657	2617	1226	174K	JULY 2
# of buffer misses (Actual I/O)	0	40	17869	83	59	18051	
Number of bytes read from DASD	0	40K	35738K	332K	472K	35.72M	
Hit ratio	N/A	100%	80%	97%	95%	90%	
Number of I/O requests	311	18726	21222	2510	1564	43K	JULY 8
# of buffer misses (Actual I/O)	13	113	3961	116	186	4389	
Number of bytes read from DASD	6.5K	113K	7922K	472K	1488K	9.76M	
Hit ratio	96%	99%	81%	95%	88%	90%	

Figure 5.4: Simulation of new buffer setup.

This would increase the number of 2K buffers without increasing the total amount of memory allocated to buffers, i.e., 249K bytes. Figure 5.4 shows the results produced when the simulator was run with the proposed buffer allocations.

Figure 5.5 compares the results of the simulation when the original buffer setup is assumed (Figure 5.4), compared with the proposed setup. The comparison shows no reduction in hit ratios, which is rather strange considering that the number of 2K buffers was more than doubled. This shows that had the changes been implemented, there would have been no noticeable change in the performance of the system.

	System	Hit Ratio of the Original Simulation	Hit Ratio of simulation with new Buffers
June 29	TESTEM	90%	90%
July 02	TESTEM	90%	90%
July 08	TESTEM	90%	90%

Figure 5.5: Comparison of two simulations

The unexpected results led us to try to find the knees of the hit ratio curves as a function of the number of buffers allocated to each pool. Figures 5.6 to 5.10 show the curves for the different sets of buffers on the three days, with the original allocations being indicated by the broken vertical lines. The data show that for this CICS system, and with the demonstrated load, the knee of the curve is passed at:

1) 10 buffers for .5K buffers.
2) 2 buffers for 1K buffers.
3) 3 buffers for 2K buffers.
4) 10 buffers for 4K buffers.
5) 15 buffers for 8K buffers.

5.3 Explanation of Results

At this point we decided that the pattern of activity in the 2K buffer would be analyzed so as to explain the apparent 80% hit ratio limit. We conjectured that the access pattern was such that a record was being accessed approximately 5 times and then never accessed again, thus the 20% miss ratio. However, when we looked at the data from 06/29, our theory was not substantiated. It turned out that of the 18,687 record accesses, only 427 different records were read,

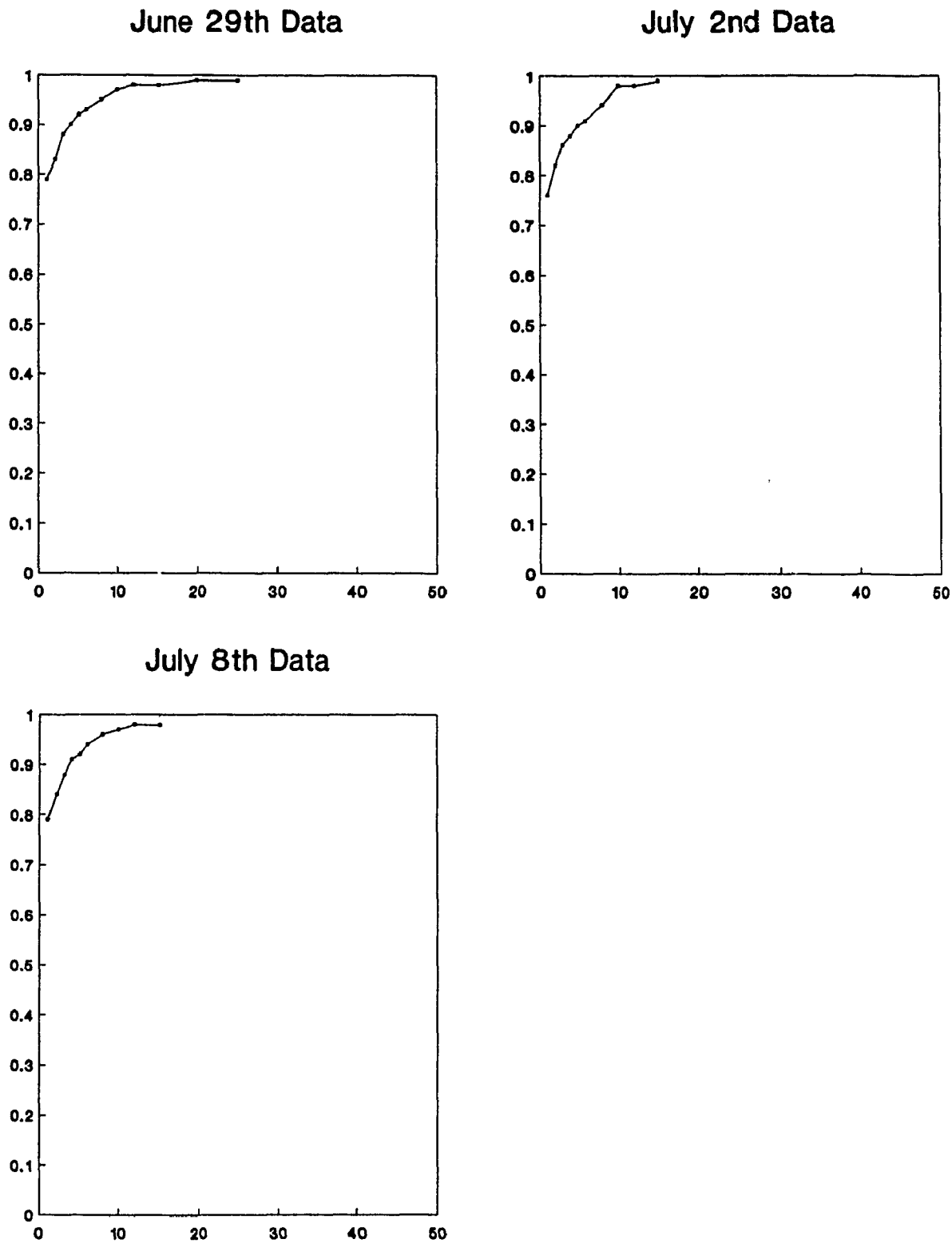


FIGURE 5.6 : HIT RATIOS FOR .5K BUFFERS

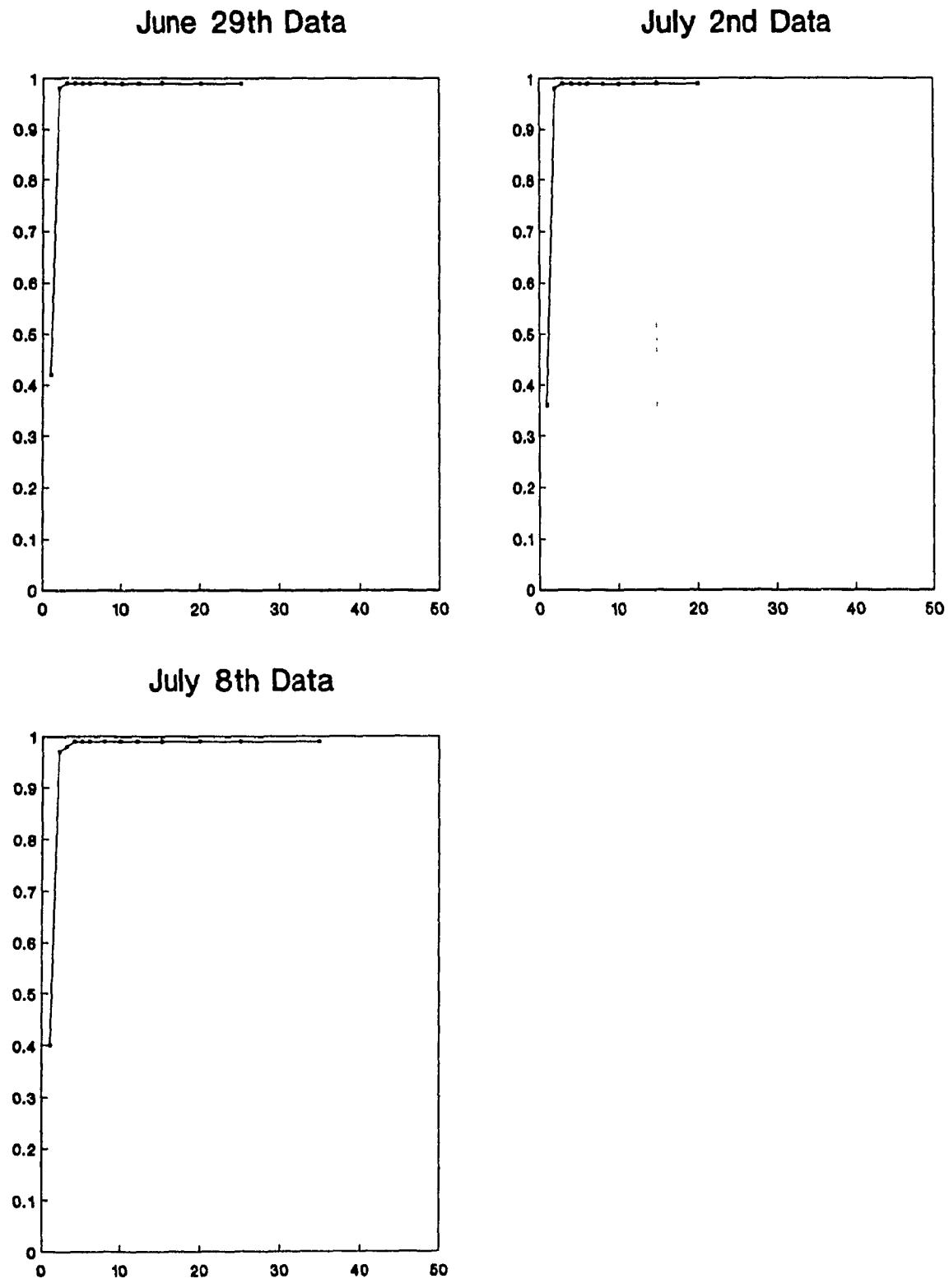


FIGURE 5.7 : HIT RATIOS FOR 1K BUFFERS

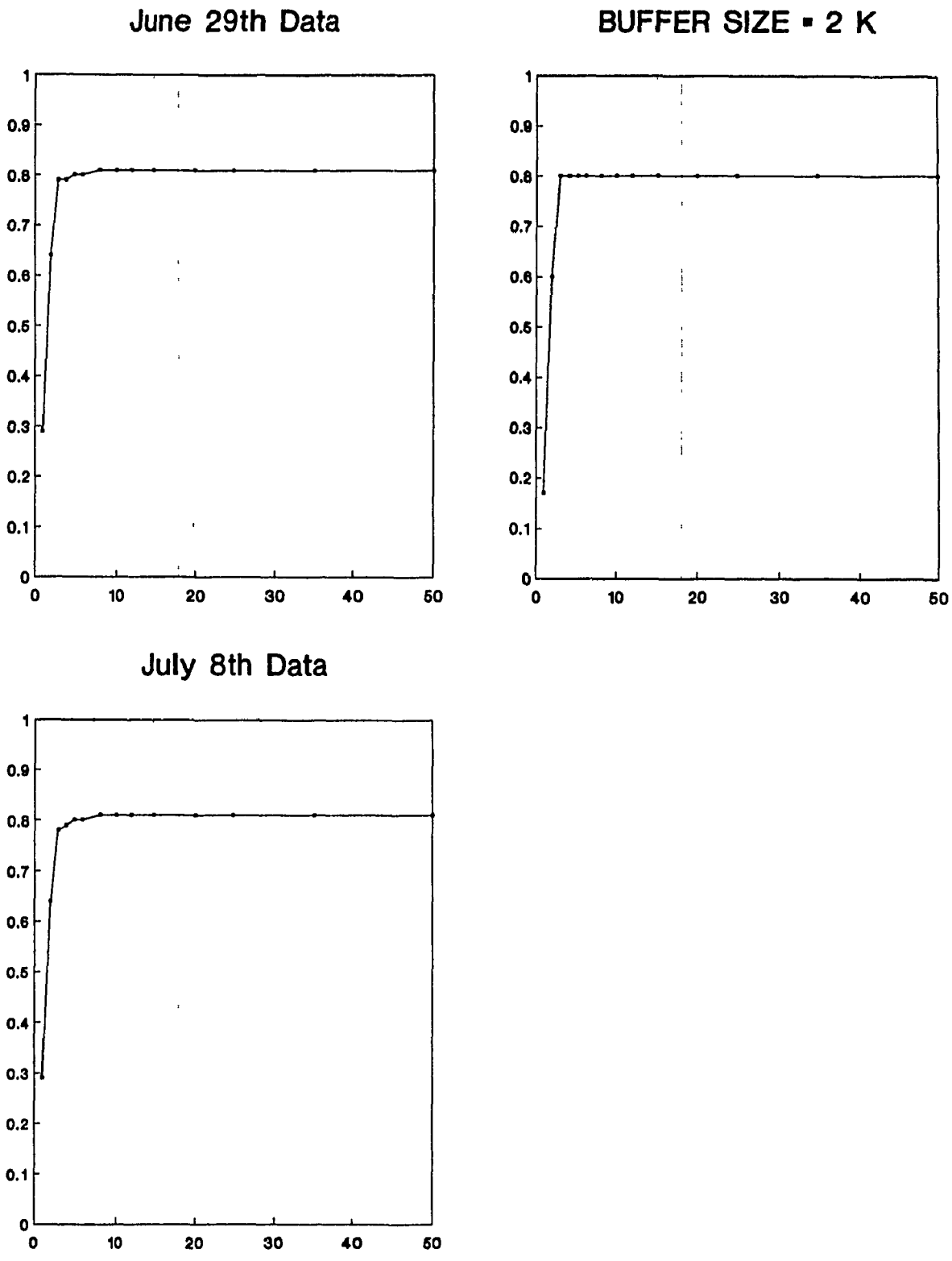


FIGURE 5.8 : HIT RATIOS FOR 2K BUFFERS

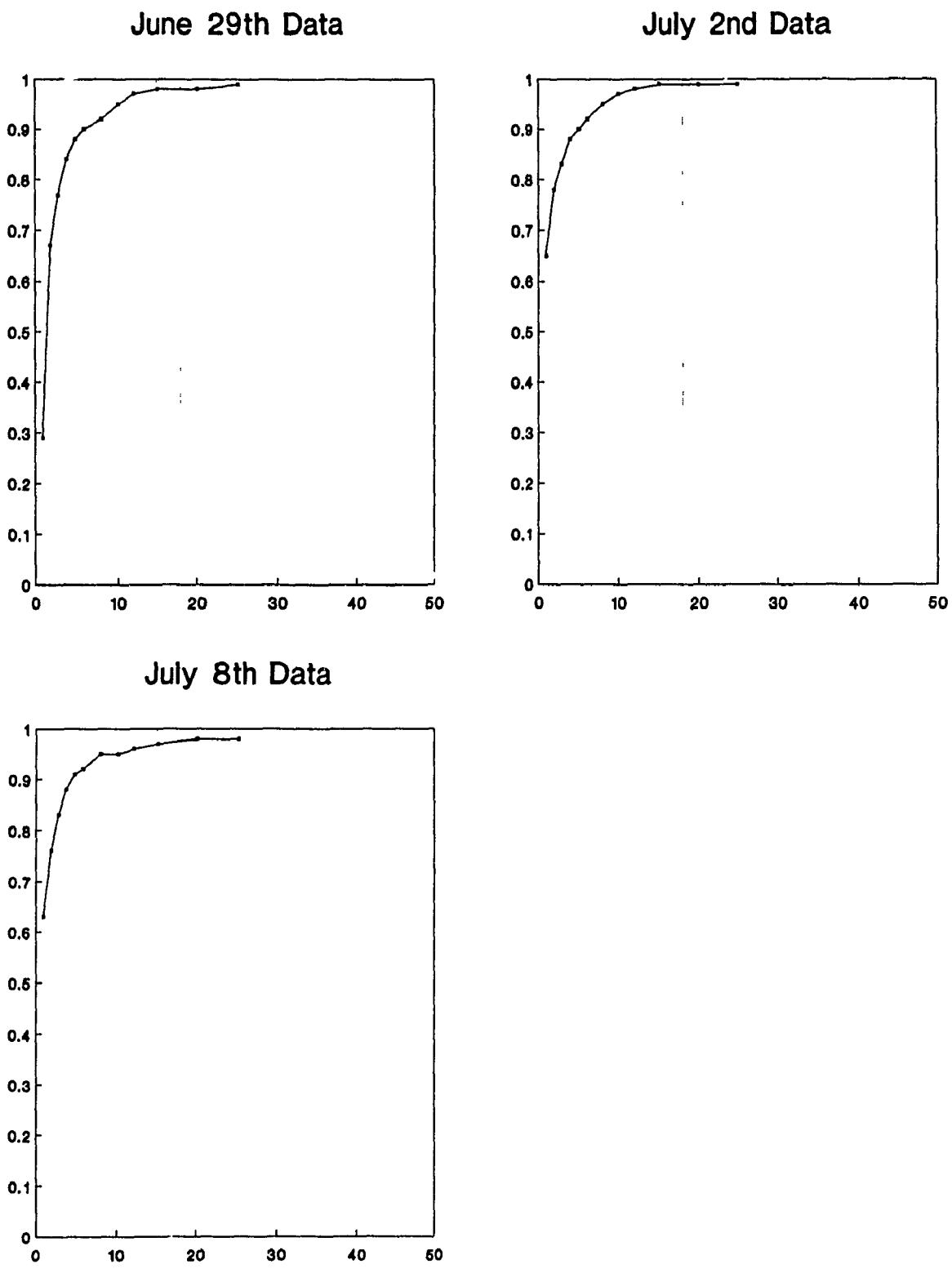


FIGURE 5.9 : HIT RATIOS FOR 4K BUFFERS

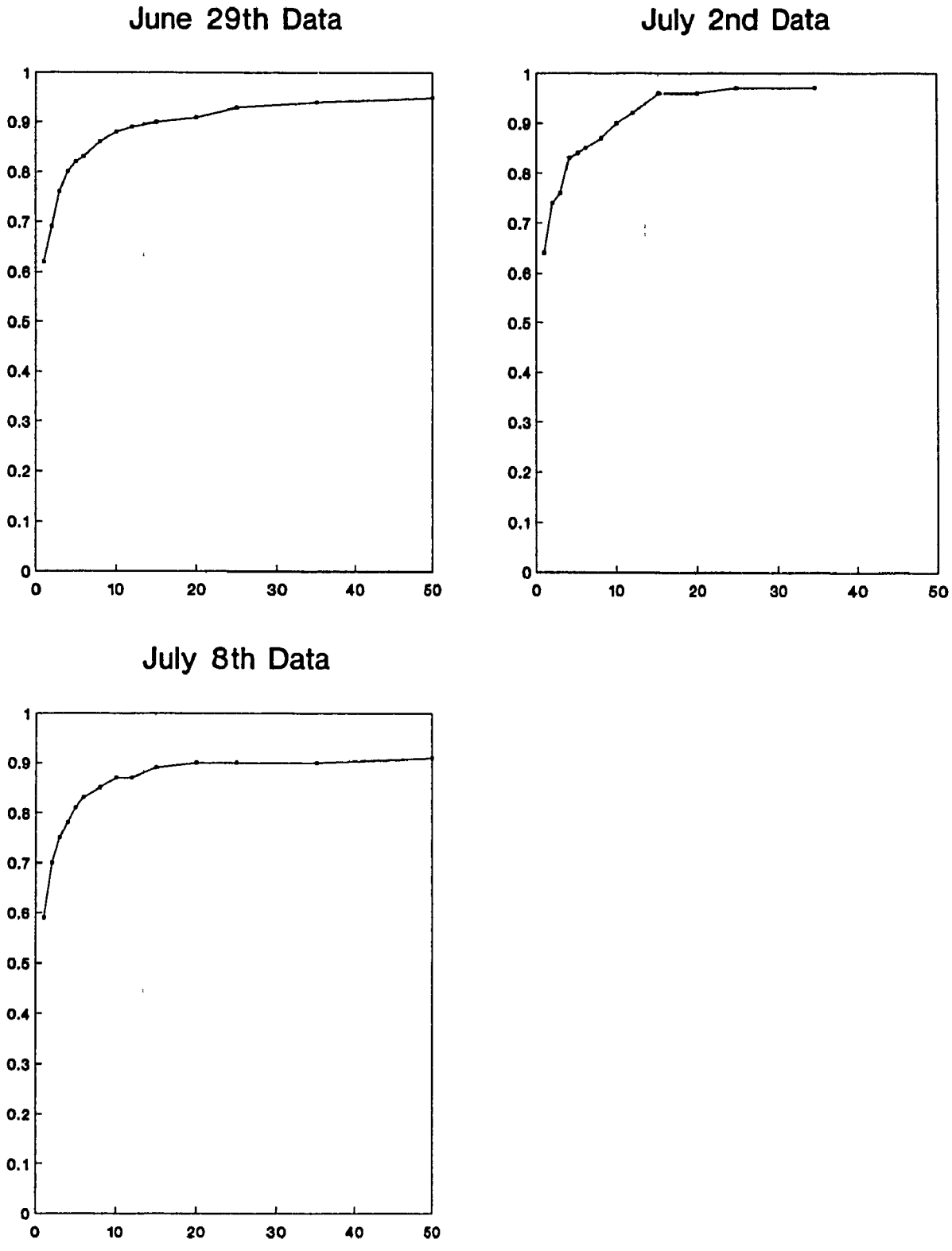


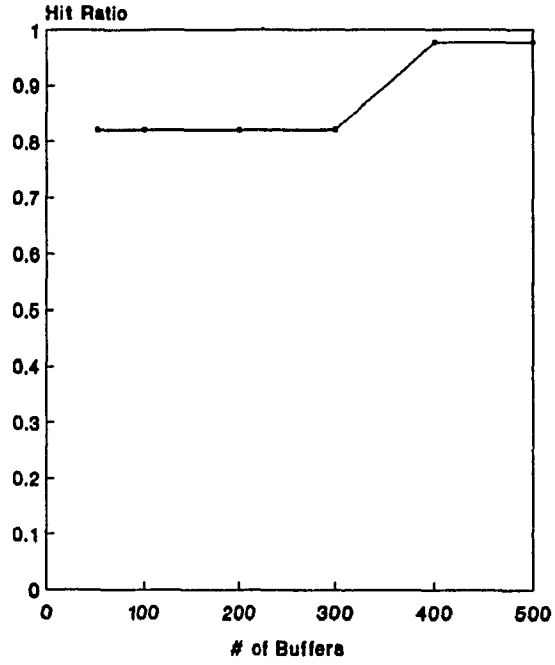
FIGURE 5.10 : HIT RATIOS FOR 8K BUFFERS

i.e., the other 18,260 accesses were of records that had already been accessed during the test. This indicated that a 97% hit ratio was achievable, albeit with an unreasonable number of buffers (427). This clearly indicated that our assumption of an 80% limit on hit ratios was not correct.

To verify this, simulations were carried out at larger buffer sizes, the results of this simulation are shown in Figure 5.11. It turned out that being a test system, certain transactions were being executed, following which the programmer might apply changes to the program (i.e., recompile), and then reexecute. Thus the re-referencing of data occurs after relatively long periods of time.

In short, the data tend to be accessed about 5 times in close proximity, after which the data are not likely to be accessed again for a very long time.

June 29th Data



July 2nd Data

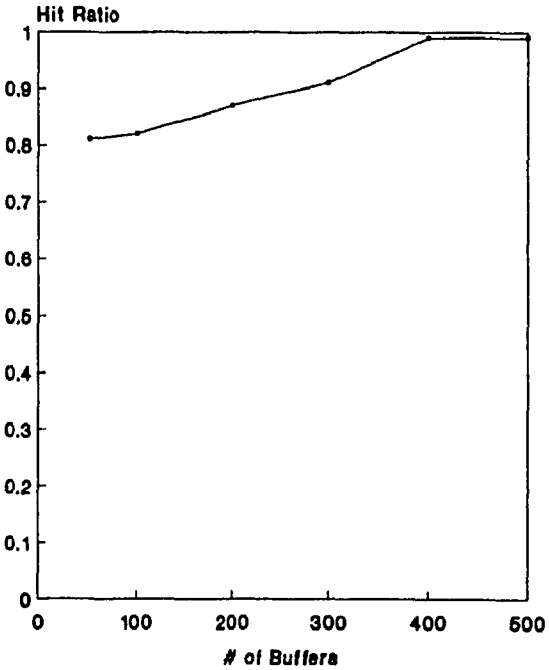


FIGURE 5.11 : HIT RATIOS FOR 2K BUFFERS

6. Improving the Simulator

The initial results (presented in Section 5) indicated that the simulator could be used to accurately predict the Hit Ratios for an Electronic-Mailing development region (TESTEM). However, it remained to be seen how well CLBS would fare in other systems. CLBS was therefore executed in two other test (development) systems, and then finally tested on two production systems. The two production systems were diverse, the first system (PRODEM) being an IBM developed Electronic Mail System (production version of TESTEM), while the second (PRODFIN) was a financial system developed by a supplier specializing in financial applications. The findings of these runs are summarized in this section.

6.1 Testing on Multiple Development Systems.

The copy of CLBS used in the tests described in section 5, was a CICS 1.6 version. However, CICS systems had since been upgraded to CICS 1.7 software, therefore, some major modifications were required to CLBS to permit it to function with the new software. There were also some minor

modifications which were implemented, e.g., the requirement to reach Steady State before commencing the simulation was dropped because it was found that this requirement did not add significantly to the accuracy of the simulation.

	System	Hit Ratio of actual run	Hit Ratio of simulation
Jan. 17	TESTD1	75%	94%
Jan. 22	TESTD1	44%	94%
Jan. 27	TESTD1	83%	93%
Jan. 16	TESTD2	26%	100%
Jan. 24	TESTD2	29%	99%
Jan. 28	TESTD2	36%	99%

Figure 6.1: Accuracy of the modified simulator.

The results of these new tests (Figure 6.1) differed from those of the previous tests (Figure 5.3) in that the new hit ratios (actuals) were much lower, even though CLBS predicted that much higher hit ratios should have been attained. To find the reason behind this problem, the data gathered by CLBS were dumped and analyzed. The analysis showed that there was one file which was causing the anomality and it was

found that the file in question was defined as shareoption 4 (SHR4). A record in a file defined as SHR4 is always read from DASD even if a copy of the record already exists in a buffer. SHR4 is used to provide integrity for files shared by multiple systems.

In SHR4 files, a request for a record will be read from DASD even if the record is already in a buffer. This is done because there is no guarantee that the record in the buffer is still valid, the record on DASD might have been modified by another system thus invalidating the copy in the buffer [WIL90].

It is strongly recommended [CAN87, WIL90] that SHR4 files not be placed in the LSR pool. However, in the conversion of the CICS systems from the 1.6 to the 1.7 level, the SHR4 files had been erroneously added. Figure 6.2 shows simulation results when CLBS was modified to accurately simulate SHR4 file handling, i.e., to read records from DASD even if they are already in a buffer. Figure 6.2 shows that with the modification, CLBS was again able to accurately simulate Hit Ratios.

	System	Hit Ratio of actual run	Hit Ratio of simulation
Jan. 17	TESTD1	75%	76%
Jan. 22	TESTD1	44%	44%
Jan. 27	TESTD1	83%	83%
Jan. 16	TESTD2	26%	25%
Jan. 24	TESTD2	29%	30%
Jan. 28	TESTD2	36%	36%

Figure 6.2: Accuracy of the simulator.

Next, CLBS was used to predict the effect of removing the SHR4 file from the buffers. Figure 6.3 shows that the hit ratios of the TESTD1 system were expected to average 91%, while the hit ratios of the TESTD2 system were expected to average 94%. To substantiate this, the SHR4 file was removed from the LSR pool and CLBS was rerun to gather data from the two test systems. Figure 6.4 shows the results of these tests, it can be seen that the actual hit ratios were very close to the expected (simulated) ranges. In TESTD1 the average hit ratio was expected to move from 68% to 91%, and a 92% average was achieved. In TESTD2 the average hit ratio was expected to move from 30% to 94%, and a 90% average was achieved.

	Hit Ratio of simulation
Jan 17	94%
Jan 22	86%
Jan 27	93%
Average	91%

a) System = TESTD1

	Hit Ratio of simulation
Jan 16	95%
Jan 24	89%
Jan 28	97%
Average	94%

b) System = TESTD2

Figure 6.3: Hit Ratios Predicted had SHR4 files been Excluded from LSR.

	Hit Ratio
Apr 09	91%
Apr 10	94%
Apr 17	92%
Apr 24	92%
Apr 26	89%
Average	92%

a) System = TESTD1

	Hit Ratio
Apr 12	88%
Apr 25	90%
Apr 27	91%
Average	90%

b) System = TESTD2

Figure 6.4: Actual Hit Ratios when SHR4 files were Excluded from LSR.

6.2 Simulation of PRODFIN

As discussed in the introduction to this section the PRODFIN system is a production online financial system running at CN. The results of 5 simulations are displayed in Figure 6.5, these results again show that CLBS was able to simulate hit ratios with high accuracy. We next tried to use the details of the runs (Figure 6.6) to find a way of improving the buffer setup.

The nine buffers allocated to the 16K pool jumped at us as obvious candidates for improvement. 144K of storage were being allocated to accommodate less than 1,000 I/O requests in a day. The 16K buffers were reduced from 9 to 3 (minimum) buffers. No measurements were taken after the change, because this recommendation would have been obvious even if CLBS had not been used.

	System	Hit Ratio of actual run	Hit Ratio of simulation
Feb. 26	PRODFIN	71%	68%
Feb. 28	PRODFIN	97%	97%
May 01	PRODFIN	74%	71%
May 02	PRODFIN	76%	72%
May 03	PRODFIN	80%	77%

Figure 6.5: Accuracy of the simulator with PRODFIN.

Buffers assigned during the runs								
Buffer sizes	512	1024	2048	4096	8192	16K	TOT	
# of buffers	13	16	30	50	6	9		
# of bytes used for buffer	6K	16K	60K	200K	48K	144K	474K	
Number of I/O requests	1K	6K	7K	15K	3K	0	31K	
# of misses (Act. I/O)	8	2652	1889	5158	444	0	3552	FEB
# of bytes read from DASD	0M	3M	4M	20M	3M	0	30M	26
Hit ratio	99%	54%	79%	66%	82%	N/A	68%	
Number of I/O requests	6K	28K	47K	84K	15K	1K	181K	
# of misses (Act. I/O)	60	9540	10514	32152	1133	4	53403	MAY
# of bytes read from DASD	0M	9M	21M	126M	9M	0M	164M	01
Hit ratio	99%	66%	78%	63%	93%	100%	71%	
Number of I/O requests	6K	34K	76K	123K	19K	1K	259K	
# of misses (Act. I/O)	47	11690	15355	45587	1996	7	74682	MAY
# of bytes read from DASD	0M	11M	30M	178M	16M	0M	235M	02
Hit ratio	99%	67%	80%	64%	90%	100%	72%	

Figure 6.6: Results from PRODFIN System.

Next we looked at the 512 pool which also had a very high hit ratio (99%) but considering that only 6K of storage were allocated to this pool, it was decided not to change this buffer pool.

Finally, a set of CLBS simulations were carried out for the 4K buffer pool to try to reduce the number of bytes of data that were being transferred. Figure 6.7 shows that increasing the number of buffers from 50 to 100 (a 200K increase in memory), would have resulted in a reduction of 15% in the number of bytes transferred from DASD. It was

# of Buffers	# of misses	# of bytes transferred from DASD	Hit Ratio of simulation
3	75,851	300M	40%
10	60,729	240M	52%
15	57,252	227M	55%
30	50,528	200M	60%
50	45,587	178M	64%
100	38,359	152M	70%
150	32,783	130M	74%
250	26,134	102M	79%

Figure 6.7: CLBS results for 4K buffer pool (0502 data)

felt that the reduction in data transfers did not warrant the large increase in memory utilization, therefore the number of 4K buffers was not changed.

6.3 Simulation of PRODEM

Figure 6.8 shows three simulations which were run against the PRODEM system and again it shows that CLBS was able to predict hit ratios with high accuracy in different CICS systems. Figure 6.9 shows the details of three runs, it can be seen that there were two buffer pools (#1 and #2) allocated. This was done by the system programmers to isolate a heavily used file, by giving it its own pool (#2). Figure 6.9 shows that most of the data transfers came from the 8K (#1) pool of buffers.

	System	Hit Ratio of actual run	Hit Ratio of simulation
May 04	PRODEM	88%	89%
May 06	PRODEM	92%	92%
May 07	PRODEM	92%	92%

Figure 6.8: Accuracy of the simulator with PRODEM.

Buffers assigned during the runs								
Buffer Pool # Buffer sizes	1 1024	1 2048	1 4096	1 8192	2 2048	2 8192	TOT	
# of buffers	20	8	34	16	28	28		
# of bytes used for buffer	20K	16K	136K	128K	56K	224K	580K	
Number of I/O requests	78K	18K	32K	78K	34K	34K	375K	
# of misses (Act. I/O)	1389	120	4259	13455	3751	7502	30476	MAY 04
# of bytes read from DASD	1M	0M	17M	105M	8M	62M	193M	
Hit ratio	98%	99%	87%	83%	89%	77%	89%	
Number of I/O requests	87K	19K	35K	87K	35K	82K	356K	
# of misses (Act. I/O)	1632	201	4477	13413	3681	8923	32327	MAY 06
# of bytes read from DASD	2M	0M	17M	13M	7M	70M	136M	
Hit ratio	98%	99%	87%	85%	90%	75%	91%	
Number of I/O requests	14K	4K	7K	14K	6K	6K	51K	
# of misses (Act. I/O)	219	2	748	1569	380	1092	4010	MAY 07
# of bytes read from DASD	0M	0M	3M	13M	1M	9M	26M	
Hit ratio	98%	100%	90%	89%	94%	83%	92%	

Figure 6.9: Results from PRODEM System.

Simulations with different numbers of 8K (#1) buffers were run and the results summarized in Figure 6.10. In this case, CLBS indicated that increasing the buffers from 16 to 30 (112K bytes of storage), would have eliminated 25% of the bytes transferred from DASD to the 8K (#1) buffer pool, therefore it was decided to implement the change. Figure 6.11 shows CLBS predictions assuming that the number of buffers were increased from 16 to 30, and shows that the hit ratio was expected to average 89%.

# of Buffers	# of misses	# of bytes transferred from DASD	Hit Ratio of simulation
3	22,116	173M	72%
10	15,724	123M	80%
16	13,455	105M	83%
30	10,041	78M	87%
50	7,347	57M	91%
100	4,659	36M	94%
150	3,680	29M	95%
250	2,921	23M	96%

Figure 6.10: CLBS results for 8K (#1) buffer pool (0504 data)

	System	Actual Hit Ratio with 16 buffers	Hit Ratio of simulation with 30 buffers
May 04	PRODEM	83%	87%
May 06	PRODEM	85%	89%
May 07	PRODEM	89%	91%
Average	PRODEM	86%	89%

Figure 6.11: Prediction of hit ratios of 8K (#1) buffer with 30 buffers.

	System	Hit Ratio with 30 buffers
May 28	PRODEM	93%
May 29	PRODEM	85%
May 30	PRODEM	87%
May 31	PRODEM	92%
June 01	PRODEM	88%
Average		89%

Figure 6.12: Hit ratios when buffers were increased to 30.

Figure 6.12 shows the hit ratios which were obtained when the number of buffers were increased from 16 to 30 on the PRODEM system, as can be seen, the hit ratio averaged 89%, exactly as predicted by CLBS.

As a final test, simulations were run on the May 28-June 1 data (30 buffers), assuming that only 16 buffers had been allocated (Figure 6.13). The simulations indicated that an average of 86% should be expected. Comparing this prediction with the May 4-7 (Figure 6.11) data (86% hit ratio) where only 16 buffers were allocated, we again find that hit ratios predicted by CLBS exactly match actual results.

	System	Actual Hit Ratio with 30 buffers	Hit Ratio of simulation with 16 buffers
May 28	PRODEM	93%	90%
May 29	PRODEM	85%	81%
May 30	PRODEM	87%	82%
May 31	PRODEM	92%	89%
June 1	PRODEM	88%	86%
Average		89%	86%

Figure 6.13: Prediction of hit ratios when buffers are decreased to 16 buffers.

	System	Hit Ratio of actual run	Hit Ratio of simulation
June 29	TESTEM	89%	90%
July 02	TESTEM	89%	90%
July 08	TESTEM	89%	90%
Jan. 16	TESTD2	26%	25%
Jan. 17	TESTD1	75%	76%
Jan. 22	TESTD1	44%	44%
Jan. 24	TESTD2	29%	30%
Jan. 27	TESTD1	83%	83%
Jan. 28	TESTD2	36%	36%
Feb. 26	PRODFIN	71%	68%
Feb. 28	PRODF_N	97%	97%
Apr 09	TESTD1	91%	91%
Apr 10	TESTD1	95%	94%
Apr 12	TESTD2	87%	88%
Apr 17	TESTD1	92%	92%
Apr 24	TESTD1	92%	92%
Apr 25	TESTD2	89%	91%
Apr 26	TESTD1	90%	89%
Apr 27	TESTD2	89%	91%
May 01	PRODFIN	74%	71%
May 02	PRODFIN	76%	72%
May 03	PRODFIN	80%	77%
May 04	PRODEM	88%	89%
May 06	PRODEM	92%	92%
May 07	PRODEM	92%	92%
May 28	PRODEM	91%	91%
May 29	PRODEM	91%	91%
May 31	PRODEM	93%	93%
June 01	PRODEM	90%	90%

Figure 6.14: Actual vs Simulated hit ratios.

6.4 Overall Validity of the Model

Figure 6.14 condenses all CLBS runs into a single table, it shows that CLBS can accurately simulate LSR buffer functioning in five different CICS systems. In 29 executions of CLBS, 23 of the simulated hit ratios were within one percentage point of the actual hit ratios.

Figure 6.15 compares CLBS predicted hit ratios with actual hit ratios that were obtained when certain changes were implemented. It shows a high correlation between CLBS predicted hit ratios and actual hit ratios without having to implement these changes in a live system.

SYSTEM	Change	Predicted Using CLBS	Actual I/O Results
TESTD1	Remove SHR4 files from LSR.	91%	92%
TESTD2	Remove SHR4 files from LSR.	94%	90%
PRODEM	Increase Buffers from 16 to 30.	89%	89%
PRODEM	Decrease Buffers from 30 to 16.	86%	86%

Figure 6.15: Summary of Predicted and Actual Hit Ratios.

7. CONCLUSIONS

DASD response time improvements have lagged behind improvements achieved in CPUs, therefore buffering has been used to overcome this deficiency. However, deciding on the number of buffers to allocate is difficult because the relation between the number of buffers and the hit ratios are not obvious. A simulator (CLBS) has been presented here that allows for the quick and safe testing of different buffer configurations, before implementing the configuration in a production environment.

The simulator was first tested in a test CICS system (TESTEM) which provided Electronic Mail functions. CLBS was validated by using it to predict the hit ratios for various LSR buffers when the number of buffers of each size were unchanged. Using the data from these simulations, proposals were made for increasing the hit ratios by changing the number of buffers allocated to certain buffer pools. Surprisingly, simulations using the new proposed changes showed that the hit ratios would not have improved. Analysis showed that the data tended to be re-accessed in close proximity a number of times, after which the data are not re-accessed for a very long time. This was determined to be

due to TESTEM being a test (non-production) CICS system where programmers tested a program (period of heavy re-accesses), then stopped testing for a while to make corrections and recompile, and then tested again (records re-accessed after a long period).

Next, CLBS was upgraded to function in a CICS 1.7 environment and it was re-validated by simulating LSR buffering in two test CICS systems and two production CICS systems. Simulations of the first production system (PRODFIN) showed that an additional 200K bytes of memory would have been required to reduce the number of I/O operations by 15%. Based on these results, it was decided not to implement the changes. Simulations of the second production system (PRODEM) showed that an additional 112K bytes of memory would have been required to reduce the number of I/O operations by 25%, the changes were therefore implemented. The modified (30 buffers) system was traced and the results were validated with CLBS simulations of the unmodified (16 buffers) PRODEM. Finally, simulations with 16 buffers were run on the modified system and validated with traces of PRODEM system before being modified. In all cases the validation showed excellent accuracy.

An unexpected benefit of these simulations was the discovery that SHR4 files had been erroneously placed in LSR buffers during the conversion of CICS from the 1.6 to the 1.7 level. SHR4 files are not good candidates for LSR because they are re-read from DASD, even if a copy of the record is already resident in a buffer, thus wasting buffers. The simulator was modified to account for SHR4 files, after which it was successful in predicting hit ratios.

CLBS has been used in a variety of IBM CICS environments, and has been shown to permit accurate assessment of proposed changes to LSR buffer allocations. Although the details of the implementation are specific to the IBM CICS environment, the same model can be used to simulate any buffering system that is based on an LRU replacement algorithm.

8. REFERENCES

[ATW82] J.W. Atwood, A. MacLeod, Keh-Chiang Yu, "An Emperical Study of a CDC 844-41 Disk Subsystem", Performance Evaluation 2 (1982), pp 29-56.

[BER87] F. Bereznay, "VSAM Specification and Tuning", Proceedings of the 1987 Computer Measurement Group Conference, pp 40-45.

[BOZ88] G. Bozikian, W. Atwood, "CICS LSR Buffer Simulator (CLBS)", Proceedings of the 1988 Computer Measurement Group Conference, pp 493-503.

[CAN87] Candle Corporation, "VSAM Tuning in a CICS Environment", Candle Computer Report (April 1987). Candle Corporation, 1999 Bundy Drive Los Angeles, CA 90025.

[DAR01] E.H. Daray, "OS/VS VSAM Sharing - A Technical Discussion", IBM Manual (G320-6015).

[DEE84] D.C. Deer, "Evaluating Capacity Planning Techniques", Journal of Capacity Management Vol. 2, No. 2 (1984), pp 106-118.

[DOH01] W.J. Doherty, A.J. Thadhani, "The Economic Value of Rapid Response Time", IBM Manual (GE20-0752-0).

[IBM78] IBM Corporation, "DASD Seek Simulator", IBM Manual (SB21-2218-0).

[KLE76] L. Kleinrock, "Queueing Systems", Vol. 1, Wiley (1976).

[LOC80] C. LoCicero, "An Event-Trace Study of the Performance of the I/O Subsystem for a CDC CYBER 172 Computer", Thesis at Concordia University, Department of Computer Science (1980).

[SAL86] M.A. Salsburg, "Simulation is not a Four Letter Word", Proceedings of the 1986 Computer Measurement Group 1986 conference, pp 128-139.

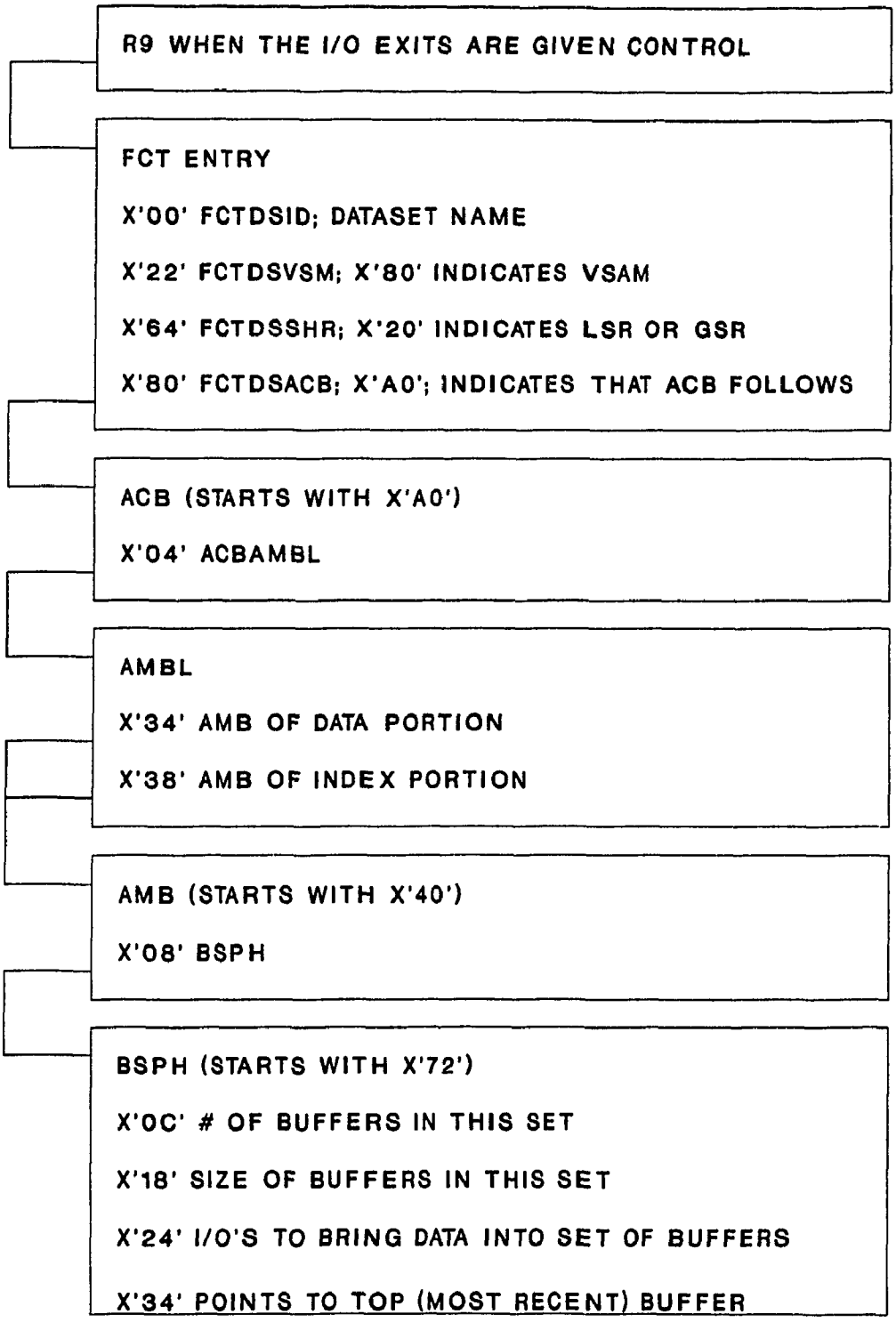
[SMI81a] A.J. Smith, "Input/Output optimization and disk architecture: a survey", Performance Evaluation 1 (1981), pp 104-117.

[SMI81b] A.J. Smith, "Optimization of I/O systems by cache disk and file migration: a summary", Performance Evaluation 1 (1981), pp 249-262.

[WIL90] E. Williams, "VSAM Tuning", Candle Computer Report (May 1990), pp 5-8. Candle Corporation, 1999 Bundy Drive Los Angeles, CA 90025.

APPENDICES

APPENDIX A
CONTROL BLOCK STRUCTURE



APPENDIX B

CLBS User's Manual

This manual shows the steps which should be taken by a CICS Systems Programmer to set up and use the CLBS environment. A knowledge of CICS and the SAS programming language is assumed. The version of CLBS presented here only works on CICS 1.7 systems, running under the MVS operating system.

Set-up of the Environment

This section defines the steps required to be taken to set up an environment under which CLBS can be executed:

a) A sequential disk dataset with LRECL=BLKSIZE=2048, must be allocated to hold the data. The size of this dataset depends on the amount of data to be collected, a 15 MB dataset will accommodate 5,000 blocks of data (i.e., 300,000 I/Os to the LSR pool). Note that the DXCWRTR program in Appendix D, will stop collecting data after 5,000 blocks of data have been collected. If this limit is changed, the DASD requirements will change proportionately.

b) Make the data set allocated in Step (a) available to the CICS region to be analysed.

c) The following entries should be added to the DCT table.

```
BUFFDATA DFHDCT TYPE=SDSCI,                    &
               DSCNAME=BUFFDATA,               &
               RECFORM=FIXUNB,                 &
               RECSIZE=2048,                   &
               BLKSIZE=2048,                   &
               BUFNO=2,                        &
               TYPEFLE=OUTPUT

BUFF     DFHDCT TYPE=EXTRA,                    &
               DESTID=BUFF,                    &
               DSCNAME=BUFFDATA
```

d) The following entries should be added to the PCT table.

```
WRTR     DFHPCT TYPE=ENTRY,                    &
               TWASIZE=0,                      &
               TRANSID=WRTR,                   &
               CLASS=LONG,                     &
               PROGRAM=DXCWRTR

POST     DFHPCT TYPE=ENTRY,                    &
               TWASIZE=0,                      &
               TRANSID=POST,                   &
               CLASS=LONG,                     &
               PROGRAM=DXCPOST
```

e) The following entries should be added to the PPT table.

```
DXCPOST  DFHPPT TYPE=ENTRY,PROGRAM=DXCPOST
DXCWRTR  DFHPPT TYPE=ENTRY,PROGRAM=DXCWRTR
```

f) The two programs in Appendices C and D, should be assembled, and the result placed in a library accessible by the CICS to be simulated (i.e., in the RPL list).

Note that the program DXCPOST must be re-entrant.

Also note that these programs will use the first 4 bytes

of the user area (CSA+200) to store the address of a getmained area. Ensure that this area is not in use at your site. If the area is in use, the programs will have to be modified to use a different portion of the user area.

g) Make sure to test CLBS in a test region first, because the code in CLBS has only been tested on systems at Canadian National.

Data Collection

Having set up the environment, starting CLBS consists of ensuring that the data set is allocated and opened to the CICS to be simulated. Once this is done, the transaction WRTR can be issued, which will enable the necessary exits, and start gathering data for the simulation. However, stopping the data collection is not as elegant, one needs to access CSA+200 for the address of the getmained area. Zapping the getmained area+12 bytes by '4' will post the Wait ECB, which will cause CLBS data gathering to terminate. Note that CLBS will automatically terminate if the limit of the number of blocks is reached, which by default is set to 5,000 blocks.

Simulation

Once the data have been gathered on a sequential dataset, they can be converted into SAS format using the program in Appendix E. It is not advisable to place SHR4 (VSAM Share Option 4) files in an LSR pool. However, if such files exist in the LSR pool at your installation, then some modifications must be made to the program in Appendix E. The statements following the comment marked by;

```
*****>>>>>>>>>   CHECK FOR SHARE OPTION 4   <<<<<<*****;
```

will have to be modified. In the sample program shown in Appendix E, two files DASDBIF and AUTOSPOL are SHR4 files which are in the LSR pool. If there had been 3 SHR4 files in the LSR pool, DADSBIF, AUTOSPOL, & WHATEVER, then the following statements would have been coded:

```
*****>>>>>>>>>   CHECK FOR SHARE OPTION 4   <<<<<<*****;
READNEW=0;
IF (FILENAME='DADSBIF' AND READIO='R')
   OR (FILENAME='AUTOSPOL' AND READIO='R')
   OR (FILENAME='WHATEVER' AND READIO='R')
        THEN READNEW=1;
*****>>>  END OF CHECK FOR SHARE OPTION 4  <<<*****;
```

Once the SAS files are created, the simulator (Appendix F) can be executed. The only statements which are to be modified by the user are the ones which execute the BUFANAL macro, identified by '*<=====' in the program. These lines may be modified, deleted, or added to as required, e.g., if one is interested in simulating the effect of increasing the 2K (2048) buffer pool to 30 buffers, the following statement would be used :

```
%BUFANAL (2048,30); .
```

POOLNUM	SZOFBUF	NUMOFBUF	NUMOFREC	NUMOFBY	NUMOFIO	ACTBYTIO	HITR	SIMULIO	SIMBYTIO	SIMULHR
1	2048	18	18686	36	3645	7290	80.5	3400	6800	82
1	2048	25	18686	36	3645	7290	80.5	3327	6654	82
1	2048	50	18686	36	3645	7290	80.5	3327	6654	82
1	2048	75	18686	36	3645	7290	80.5	3327	6654	82
1	2048	100	18686	36	3645	7290	80.5	3327	6654	82

Figure B.1: Sample Simulator Output

Output of Simulation

Figure B.1 shows the output from a simulator run. A description of the fields in Figure B.1 follows:

POOLNUM : Buffer pool number. CICS 1.7 allows the allocation of multiple pools.

SZOFBUF : Size of buffer pool being simulated.

NUMOFBUF: Number of buffers being simulated.

NUMOFREC: Number of I/O requests to this buffer encountered during the data gathering phase.

NUMOFMBY: Number of Megabytes of data that were requested by the application during the data gathering phase.

NUMOFIO : Number of physical I/Os (i.e., buffer misses) performed during the data gathering phase.

ACTBYTIO: Number of bytes transferred from disk to memory during the data gathering phase.

HITR : Hit ratio of this buffer pool during the data gathering phase.

SIMULIO : Number of physical I/Os (i.e., buffer misses) predicted by this simulation.

SIMBYTIO: Number of bytes transferred from disk to memory, predicted by this simulation.

SIMULHR : Hit ratio of this buffer pool predicted by this simulation.

Interpretation of Output

The first step is to verify the accuracy of the simulator in your environment. This is done by executing the simulator with the number of buffers that were allocated when the data was gathered, e.g., if there were 10 x 2K buffers allocated during the data gathering phase, then the first simulation that is run should use this value, i.e. %BUFANAL(2048,10). Accuracy can be weighed by comparing HITR with SIMULHR. If the difference is more than 10 percentage points for a certain buffer pool, the simulator should not be used for that buffer pool. Having established the accuracy of the simulator, new buffer allocations can be simulated, e.g., 20 x 2K, 30 x 2K, etc. In these cases, SIMULHR of different simulation runs can be used to find the optimum number of buffers to be allocated.

Example of Analysis

In the case of the data in Figure B.1, there were 18x2K buffers in the system, therefore the first step is to check the accuracy of the simulator for this specific environment,

i.e., 80.5 versus 82.2, which is within the criterion of ten percentage point mentioned above. Having established the accuracy to be acceptable, one can then compare the data from the different simulations. In the case of Figure B.1, it is obvious that adding more buffers would be of little benefit.

APPENDIX C

DXCPOST PROGRAM

```
*
* THIS PROGRAM IS THE XFCINC EXIT CALLED AFTER EVERY READ
*    AS WELL AS THE XFCOUT EXIT CALLED BEFORE EVERY WRITE
*
* REGISTER USAGE:
ZERO     EQU   0        CONTAINS ZERO FOR COMPARES
R01      EQU   1        PARAMETER LIST (FROM CICS)
INDEXAMB EQU   2        POINTER TO THE AMB OF THE INDEX
R03      EQU   3        BASE REGISTER
AREABAR  EQU   4        POINTS TO GETMAINED AREA
AREA1BAR EQU   5        POINTS TO FIRST AREA
NXTENTRY EQU   6        NEXT AVAILABLE ENTRY
R07      EQU   7        WORK REGISTER
R08      EQU   8        WORK REGISTER TO HOLD CB ADDRESSES
FCTDSBAR EQU   9        FCT ENTRY POINTER(FROM CICS)
R10      EQU   10       FWA ONLY IN XFCOUT (FROM CICS)
R11      EQU   11       VSWA ONLY IN XFCINC (FROM CICS)
R12      EQU   12       WORK REGISTER
R13      EQU   13       ADDRESS OF SAVE AREA (FROM CICS)
R14      EQU   14       PROGRAM RETURN POINT (FROM CICS)
R15      EQU   15       PROGRAM ENTRY POINT  (FROM CICS)
*************************************************************

         EJECT
         PRINT  NOGEN
         COPY   DFHFCTDS
         USING  DFHUEPAR,R01
         DFHUEXIT TYPE=EP

********************FIRST ENTRY OF FIRST AREA*************
ECBENT   DSECT
         USING   *,AREA1BAR
         DS      CL8
         DS      F
ECB      DS      F              ECB

********************FIRST ENTRY OF EACH AREA**************
FIRSTENT DSECT
         USING   *,AREABAR
COND     DS      CL8            STATUS OF THE AREA
LASTENT  DS      F              LAST UPDATED ENTRY
         DS      F              ECB (ONLY IN FIRST AREA)
```

```
************SUBSEQUENT ENTRIES OF EACH AREA**************
ENTRY    DSECT
         USING   *,NXTENTRY
FILENAME DS      CL8           FILENAME OF THE RECORD.
RBA      DS      F             RBA OF THE RECORD.
LRURBA1  DS      F             RBA OF NEXT LRU BUFF.
LRURBA2  DS      F             RBA OF NEXT LRU BUFF.
LRURBA3  DS      F             RBA OF NEXT LRU BUFF.
NUMOFRDS DS      F             NUMBER OF READS
SZOFBUF  DS      CL1           SIZE OF BUFFERS.
NUMOFBUF DS      CL1           NUMBER OF BUFFERS.
POOLNUM  DS      CL1           # OF THE POOL BEING USED
IOTYPE   DS      CL1           TYPE OF IO
*   XXX0 XXXX INDICATES INDEX IO
*   XXX1 XXXX INDICATES DATA IO
*   1XXX XXXX INDICATES SPANNED RECORD
*   0XXX XXXX INDICATES NON SPANNED RECORD
*   XX1X XXXX INDICATES A READ OPERATION
*   XX0X XXXX INDICATES A WRITE OPERATION
*   XXXX ZZZZ ZZZZ INDICATES NUMBER OF INDEX LEVELS
************************************************************
DXCPOST  CSECT
         USING *,R03                  USE REG. 3 FOR BASE
         STM   R14,R12,12(R13)        SAVE THE REGISTERS
         LR    R03,R15                LOAD THE BASE REG.
         TM    FCTDSVSM,FCTVSAMI      IS THIS A VSAM DS?
         BZ    BYE                    NO - GET OUT
         TM    FCTDSSHR,FCTSHRIM      DS'S BUFFERS SHARED?
         BZ    BYE                    NO - GET OUT
         SR    ZERO,ZERO              ZERO OUT THE REG.
         LR    INDEXAMB,ZERO          ZERO OUT THE REG.
         LR    R08,ZERO               ZERO OUT THE REG.
         L     R12,UEPCSA             LOAD POINTER TO CSA
         L     AREABAR,X'200'(R12)    LOAD ADR. OF GETMAIN
         CR    ZERO,AREABAR           ADR. OF GETMAIN=0?
         BE    BYE                    YES- GET OUT
         LR    AREA1BAR,AREABAR       LOAD AREA1BAR
         CLC   BTHARNU,COND           IS DATA IN ERROR?
         BE    BYE                    YES - GET OUT
         CLC   ARINUSE,COND           IS AREA ACTIVE?
         BE    AR1INUSE               YES -
         A     AREABAR,ARLENGTH       POINT TO NEXT AREA
         CLC   BTHARNU,COND           IS DATA IN ERROR?
         BE    BYE                    YES - GET OUT
         CLC   ARINUSE,COND           IS AREA 2 ACTIVE?
         BE    AR2INUSE               YES -
         MVC   COND,BTHARNU           NO ACTIVE AREAS -ERR
         LR    AREABAR,AREA1BAR
         MVC   COND,BTHARNU           INDICATE ERR IN BOTH
         B     BYE                         AREAS & RETURN
```

```
AR1INUSE EQU   *
AR2INUSE EQU   *
         L     NXTENTRY,LASTENT      POINT TO LAST USED
         A     NXTENTRY,ENTRYLNG     POINT TO NEXT ENTRY
         ST    NXTENTRY,LASTENT      UPDT LAST ENTRY
         L     R07,ARLENGTH          POINT PAST THIS AREA
         CR    NXTENTRY,R07          STILL IN AREA?
         BL    ARNOTFUL                YES -
         MVC   COND,ARFULL           INDICATE AREA FULL
         MVC   ECB,POST              POST THE WRITER TASK
         CR    AREABAR,AREA1BAR      USING AREA 1?
         BE    USEAREA2              YES - USE AREA 2 NOW
USEAREA1 LR    AREABAR,AREA1BAR      NO- BACK TO AREA 1
         B     ISARFULL
USEAREA2 A     AREABAR,ARLENGTH      POINT TO AREA 2
ISARFULL CLC   COND,ARFULL           IS AREA FULL?
         BNE   AREAOK                NO - ALL IS WELL
         MVC   COND,BTHARF           YES-BOTH AREAS FULL
         LR    AREABAR,AREA1BAR
         MVC   COND,BTHARF           INDICATE ERROR
         A     AREABAR,ARLENGTH         IN BOTH AREAS
         MVC   COND,BTHARF              THEN GET OUT
         B     BYE
AREAOK   EQU   *
         MVC   COND,ARINUSE
         L     NXTENTRY,LASTENT      POINT TO LAST USED
         A     NXTENTRY,ENTRYLNG     POINT TO NEXT ENTRY
         ST    NXTENTRY,LASTENT      UPDATE LAST ENTRY
ARNOTFUL EQU   *
         AR    NXTENTRY,AREABAR      ADD START TO ENTRY
         STC   ZERO,IOTYPE           ZERO OUT THE IO TYPE
         MVC   FILENAME,FCTDSID      MOVE DATA SET NAME
         MVC   POOLNUM,FCTIPOOL      SAVE POOL# IN USE
         CR    ZERO,R08              INDEX AMB TO HANDLE?
         BNE   INDXAMB               YES - HANDLE IT
DATAAMB  EQU   *
         LA    R08,FCTDSACB          R08 NOW POINTS TO ACB
         CLI   0(R08),X'A0'          IS THERE A VALID ACB?
         BNE   BADACB                NO - HANDLE ERROR
         L     R08,X'04'(R08)        R08 NOW POINTS TO AMBL
         L     INDEXAMB,X'38'(R08)   SAVE INDEX AMB ADDRESS
         OI    IOTYPE,X'10'          INDICATE DATA IO
         L     R08,X'34'(R08)        R08 POINTS TO DATA AMB
         CR    ZERO,R08              IS ADDRESS ZERO
         BE    ZAMB                  YES - HANDLE ERROR
         B     GETAMB
```

```
INDXAMB  EQU   *
         L     R07,X'14'(R08)       R07 POINTS TO AMDSB
         CLI   0(R07),X'60'         IS THIS A VALID AMDSB?
         BNE   BADAMDSB             NO - HANDLE ERROR
         CLI   X'39'(R07),X'10'     HANDLE COND OF # >X'F'
         BL    SETLEVEL
         MVI   IOTYPE,X'0F'         # OF INDX LVL> X'F'
         B     GETAMB
SETLEVEL EQU   *
         MVC   IOTYPE,X'39'(R07)  STORE # OF INDEX LEVELS
GETAMB   EQU   *
         CLI   0(R08),X'40'       VALID AMB
         BNE   BADAMB             NO - HANDLE ERROR
         L     R08,8(R08)         R08 NOW POINTS TO BSPH
         CR    ZERO,R08           IS ADDRESS ZERO
         BE    ZBSPH              YES - HANDLE ERROR
         CLI   0(R08),X'72'       VALID BSPH
         BNE   BADBSPH            NO - HANDLE ERROR
         L     R07,X'18'(R08)     R07=SIZE OF BUFFERS
         SRA   R07,9              DEVIDE R07 BY 512
         STC   R07,SZOFBUF
         MVC   NUMOFRDS,X'24'(R08)    NUMBER OF READS
         MVC   NUMOFBUF,X'0D'(R08)    # OF BUFFERS
         L     R08,X'34'(R08)         R08 POINTS TO BUFC
         CR    ZERO,R08               IS ADDRESS ZERO
         BE    ZBUFC                  YES - HANDLE ERROR
         TM    1(R08),X'40'           IS RECORD SPANNED?
         BZ    NOTSPAN
         OI    IOTYPE,X'80'           INDICATE SPANNED
NOTSPAN  EQU   *
         L     R07,UEPEXN         ADDRESS OF EXIT NUMBER
         CLI   0(R07),XFCOUT      IS OPERATION A WRITE?
         BE    WRITEOP
         OI    IOTYPE,X'20'       INDICATE A READ OP.
WRITEOP  EQU   *
         MVC   RBA,40(R08)        SAVE READ RBA OF RECORD
         L     R07,60(R08)        R07 -> NEXT LRU BUFFER
         CR    R07,ZERO
         BE    INDEX
         MVC   LRURBA1,40(R07)
         L     R07,60(R07)        R07 -> NEXT LRU BUFFER
         CR    R07,ZERO
         BE    INDEX
         MVC   LRURBA2,40(R07)
         L     R07,60(R07)        R07 -> NEXT LRU BUFFER
         CR    R07,ZERO
         BE    INDEX
         MVC   LRURBA3,40(R07)
```

```
INDEX    EQU   *
         LR    R08,INDEXAMB     LOAD INDEX AMB
         LR    INDEXAMB,ZERO    ZERO OUT THE REGISTER
         CR    ZERO,R08         ANOTHER INDEX TO HANDLE?
         BNE   AR1INUSE         YES GO AND HANDLE IT
         SPACE 2
BYE      EQU   *                NORMAL EXIT
         RETURN (14,12)
BADACB   MVC   FILENAME,=C'0BADACB'
         B     BYE
ZAMB     MVC   FILENAME,=C'0ZAMB'
         B     BYE
BADAMB   MVC   FILENAME,=C'0BADAMB'
         B     BYE
BADAMDSB MVC   FILENAME,=C'0BADAMDS'
         B     BYE
ZBSPH    MVC   FILENAME,=C'0ZBSPH'
         B     BYE
BADBSPH  MVC   FILENAME,=C'0BADBSPH'
         B     BYE
ZBUFC    MVC   FILENAME,=C'0ZBUFC'
         B     BYE
         EJECT
*************************************************************
* CONSTANTS AND LITERALS
*************************************************************
*
************POSSIBLE VALUES FOR COND       **************
BTHARNU  DC  CL8'0BTHARNU'   NEITHER AREA IS BEING USED
BTHARF   DC  CL8'0BTHARF'    BOTH AREAS ARE FULL
BTHARNF  DC  CL8'0BTHARNF'   NEITHER AREA IS FULL
ARINUSE  DC  CL8'0ARINUSE'   AREA IS IN USE
ARFULL   DC  CL8'0ARFULL'    AREA IS FULL I.E. WRITE OUT
*************************************************************
DECIMAL  DS    D
POST     DC    XL4'40008000'
*
ENTRYLNG DC    XL4'20'       THE LENGTH OF EACH ENTRY
ARLENGTH DC    XL4'7E0'      THE LENGTH OF EACH AREA
         SPACE 5
         LTORG
         SPACE 5
         END   DXCPOST
```

APPENDIX D

DXCWRTR PROGRAM

```
*
* THIS PROGRAM WRITES OUT THE READ AND WRITE EXIT AREAS
*
* REGISTER USAGE:
R00       EQU   0
R01       EQU   1
R02       EQU   2
R03       EQU   3        BASE REGISTER
AREABAR   EQU   4        POINTS TO GETMAINED AREA
AREA1BAR  EQU   5        POINTS TO FIRST AREA
R06       EQU   6        COUNT OF NUMBER OF AREAS WRITTEN
R07       EQU   7        WORK REGISTER
COUNTER   EQU   8        COUNTER
CSACWAR   EQU   9        POINTS TO CWA
R10       EQU   10
R11       EQU   11
R12       EQU   12       ADDRESS OF TCA
R13       EQU   13       SAVE REGISTER
R14       EQU   14       RESERVED - PROGRAM RETURN POINT
R15       EQU   15       RESERVED - PROGRAM ENTRY POINT
*************************************************************
********************FIRFIRST AREA****************************
ECBENT    DSECT
          USING   *,AREA1BAR
          DS      CL8
          DS      F
ECB       DS      F              ECB
********************FIRST ENTRY OF EACH AREA**************
FIRSTENT  DSECT
          USING   *,AREABAR
COND      DS      CL8            STATUS OF THE AREA
LASTENT   DS      F              LAST UPDATED ENTRY
          DS      F              ECB (ONLY IN FIRST AREA)
BLOCKNUM  DS      F              BLOCK NUMBER
*************************************************************
          EJECT
DXCWRTR   CSECT
*        ******   GET ADDRESS OF CWA ******
          EXEC CICS HANDLE CONDITION ERROR(BYE)
          EXEC CICS ADDRESS CWA(CSACWAR)
          SR    R06,R06              ZERO OUT THE COUNTER
```

```
*        ******   GETMAIN AREAS  ******
         EXEC CICS GETMAIN SET(AREABAR) LENGTH(GMLENGTH) X
                   INITIMG(X'00')
         EXEC CICS HANDLE CONDITION ERROR(FREE)
         ST    AREABAR,0(CSACWAR)       STORE ADDR IN CWA
         LR    AREA1BAR,AREABAR         KEEP ADDRESS
         MVC   COND,ARINUSE             AREA CAN BE USED
         ST     R06,BLOCKNUM            SAVE COUNTER
         A      AREABAR,ARLENGTH        POINT TO AREA 2
*        ******   ENABLE THE EXITS   ******
         EXEC   CICS ENABLE PROGRAM('DXCPOST')           X
                     EXIT('XFCOUT') START
         EXEC CICS HANDLE CONDITION ERROR(DISABLE2)
         EXEC   CICS ENABLE PROGRAM('DXCPOST')           X
                     EXIT('XFCINC') START
         EXEC CICS HANDLE CONDITION ERROR(DISABLE)
         SPACE 2
LOOP     EQU    *
         LA     R06,1(R06)          ADD TO COUNTER
         ST     R06,BLOCKNUM        ENTER COUNTER
         LA     R07,ECB
         EXEC   CICS WAIT EVENT ECADDR(R07)
         LR     AREABAR,AREA1BAR
         CLC    COND,ARFULL         IS AREA 1 FULL?
         BE     WRITEOUT            YES -GO AND WRITE OUT
         A      AREABAR,ARLENGTH    POINT TO AREA 2
         CLC    COND,ARFULL         IS AREA 2 FULL?
         BE     WRITEOUT           YES -GO AND WRITE OUT
         MVC    COND,BTHARNF       NO -INDICATE THAT NO
         LR     AREABAR,AREA1BAR          AREA IS FULL
         MVC    COND,BTHARNF
         L      R06,NUMOFREC       SOMETHING IS WRONG,STOP
WRITEOUT EQU    *
         EXEC CICS WRITEQ TD QUEUE('BUFF') FROM(COND)    X
               LENGTH(AREALEN)
         SR     R07,R07
         ST     R07,LASTENT
         ST     R07,ECB
         ST     R07,COND            ZERO CONDITION OF AREA
         C      R06,NUMOFREC        IS IT TIME TO STOP?
         BL     LOOP
         SPACE 2
DISABLE  EQU    *
         EXEC CICS HANDLE CONDITION ERROR(DISABLE2)
         EXEC   CICS DISABLE PROGRAM('DXCPOST')          X
                 EXIT('XFCINC') STOP
DISABLE2 EQU    *
         EXEC CICS HANDLE CONDITION ERROR(FREE)
         EXEC   CICS DISABLE PROGRAM('DXCPOST')          X
                             EXIT('XFCOUT') STOP
```

```
FREE     SR     R07,R07
         ST     R07,0(CSACWAR)
         EXEC CICS HANDLE CONDITION ERROR(BYE)
         EXEC CICS FREEMAIN DATA(FIRSTENT)
BYE      EXEC CICS RETURN
*-------------------------------------------------------------*
* CONSTANTS AND LITERALS
*-------------------------------------------------------------*
************POSSIBLE VALUES FOR COND       **************
BTHARNU  DC     CL8'0BTHARNU'     NO AREA IS BEING USED
BTHARF   DC     CL8'0BTHARF'      BOTH AREAS ARE FULL
BTHARNF  DC     CL8'0BTHARNF'     NEITHER AREA IS FULL
ARINUSE  DC     CL8'0ARINUSE'     AREA IS IN USE
ARFULL   DC     CL8'0ARFULL'      AREA IS FULL, WRITE OUT
*-------------------------------------------------------------*
*
         DS     0F
ARLENGTH DC     XL4'7E0'          THE LENGTH OF EACH AREA
GMLENGTH DC     XL2'1200'         GETMAIN LENGTH
AREALEN  DC     XL2'800'          THE LENGTH OF EACH ENTRY
NUMOFREC DC     F'5000'           NUM OF WRITES TO RECORD
         SPACE 5
         LTORG
         SPACE 5
         END   DXCWRTR
```

APPENDIX E

CONVERSION OF DATA TO SAS FORMAT

```
DATA
  DATAXXXX   (DROP = FIRSTENT COND ENT IOTYPE);
  FORMAT SPANNED READIO INDEXIO $1.;
  IF DATAEND THEN STOP;
  INFILE DATA END=DATAEND;
  INPUT @1   COND      $8.
        @1   FIRSTENT $32.  @;
  BLOCK=0; RECORD+1; ENT=1;
  IF  COND ^= '0ARFULL' THEN DO;
  ************************************************************
  ***** PROCESS THE BLOCK OF DATA     ********************
  ************************************************************;
  NEXTREC:
  ENT=ENT+32;
  BLOCK=BLOCK+1;
  INPUT @ENT FILENAME $8.
             RBA      PIB4.
             LRURBA1  PIB4.
             LRURBA2  PIB4.
             LRURBA3  PIB4.
             NUMOFRDS PIB4.
             SZOFBUF  PIB1.
             NUMOFBUF PIB1.
             POOLNUM  PIB1.
             IOTYPE   PIB1.@;
  SZOFBUF=SZOFBUF*512;
  IF IOTYPE>=80X THEN DO SPANNED='S';IOTYPE=IOTYPE-80X;END;
                    ELSE SPANNED='N';
  IF IOTYPE>=20X THEN DO READIO='R';IOTYPE=IOTYPE-20X;END;
                    ELSE READIO='W';
  IF IOTYPE>=10X THEN DO INDEXIO='D';IOTYPE=IOTYPE-10X;END;
                    ELSE INDEXIO='I';
  INDEXLVL=IOTYPE;
  *****>>>>>>>>>   CHECK FOR SHARE OPTION 4    <<<<<<*****;
  READNEW=0;
  IF (FILENAME='DADSBIF' AND READIO='R')
     OR (FILENAME='AUTOSPOL' AND READIO='R')
          THEN READNEW=1;
  *****>>>  END OF CHECK FOR SHARE OPTION 4  <<<*****;
  OUTPUT DATAXXXX;
  IF BLOCK<62 THEN GOTO NEXTREC;
  RETURN ;
```

```
OPTION LS=80;
PROC SORT DATA=DATAXXXX OUT=PERM.DATAXXXX;
   BY POOLNUM SZOFBUF RECORD BLOCK;
RUN;
DATA RECORDS (KEEP=POOLNUM SZOFBUF NUMOFREC NUMOFBUF);
  SET PERM.DATAXXXX; BY POOLNUM SZOFBUF;
  NUMOFREC+1;
  IF LAST.SZOFBUF THEN DO;OUTPUT;NUMOFREC=0;END;
PROC PRINT DATA=RECORDS;
```

APPENDIX F

SIMULATOR PROGRAM

```
DATA R512   (DROP=BLOCK RECORD NUMOFBUF)
     R1024  (DROP=BLOCK RECORD NUMOFBUF)
     R2048  (DROP=BLOCK RECORD NUMOFBUF)
     R4096  (DROP=BLOCK RECORD NUMOFBUF)
     R8192  (DROP=BLOCK RECORD NUMOFBUF)
     R12288 (DROP=BLOCK RECORD NUMOFBUF)
     R16384 (DROP=BLOCK RECORD NUMOFBUF)
     R20480 (DROP=BLOCK RECORD NUMOFBUF)
     R32768 (DROP=BLOCK RECORD NUMOFBUF);
  SET PERM.DATAXXXX;
  POOL=1;     * SET THIS VALUE TO DESIRED LSR POOL NUMBER;
  IF POOLNUM=POOL AND SZOFBUF=512 THEN OUTPUT R512;
  IF POOLNUM=POOL AND SZOFBUF=1024 THEN OUTPUT R1024;
  IF POOLNUM=POOL AND SZOFBUF=2048 THEN OUTPUT R2048;
  IF POOLNUM=POOL AND SZOFBUF=4096 THEN OUTPUT R4096;
  IF POOLNUM=POOL AND SZOFBUF=8192 THEN OUTPUT R8192;
  IF POOLNUM=POOL AND SZOFBUF=12288 THEN OUTPUT R12288;
  IF POOLNUM=POOL AND SZOFBUF=16384 THEN OUTPUT R16384;
  IF POOLNUM=POOL AND SZOFBUF=20480 THEN OUTPUT R20480;
  IF POOLNUM=POOL AND SZOFBUF=32768 THEN OUTPUT R32768;
RUN;
* OPTIONS MACROGEN MPRINT;
OPTIONS LINESIZE=132;
%MACRO BUFANAL(BUFSZ,NUMB);
%LET NUMBPLUS=%EVAL(&NUMB+1);
RUN;
DATA S&BUFSZ(KEEP = POOLNUM SZOFBUF NUMOFBUF SIMULIO
                    NUMOFIO NUMOFMBY NUMOFREC ACTBYTIO
                    SIMBYTIO HITR SIMULHR)
  SET R&BUFSZ END=NOMORE;
  FORMAT ACTBYTIO SIMBYTIO 8.;
  FORMAT NUMOFMBY HITR SIMULHR 8.2;
  RETAIN AFIL1-AFIL&NUMBPLUS '        ';
  RETAIN NUMOFBUF &NUMB;
  RETAIN PNUMOFRD ONUMOFRD ARBA1-ARBA&NUMBPLUS 0;
  ARRAY AFIL{&NUMBPLUS} $8  AFIL1-AFIL&NUMBPLUS;
  ARRAY ARBA{&NUMBPLUS} $8  ARBA1-ARBA&NUMBPLUS;
```

```
  IF NOMORE THEN DO;                          *** TERMINATION ;
        IF ONUMOFRD=0 THEN NUMOFIO=0;
                      ELSE NUMOFIO=PNUMOFRD-ONUMOFRD;
        ACTBYTIO=ROUND((&BUFSZ*NUMOFIO /1024),1);
        SIMBYTIO=ROUND((&BUFSZ*SIMULIO /1024),1);
        HITR     =((NUMOFREC-NUMOFIO )/NUMOFREC)*100;
        SIMULHR =((NUMOFREC-SIMULIO)/NUMOFREC)*100;
        NUMOFMBY=((SZOFBUF*NUMOFREC)/1024/1024);
        OUTPUT S&BUFSZ;
        RETURN;
        END;
  IF ONUMOFRD=0 THEN ONUMOFRD=NUMOFRDS;    *** INITIALIZE;
  PNUMOFRD=NUMOFRDS;
  NUMOFREC+1;
  RBA0=RBA;
  IF INDEXLVL=0 THEN INDEXLVL=1;
  IF INDEXLVL>4 THEN INDEXLVL=4;
  DO WHILE (INDEXLVL >0);
    IF INDEXLVL=1 THEN RBA=RBA0;
       ELSE IF INDEXLVL=2 THEN RBA=LRURBA1;
       ELSE IF INDEXLVL=3 THEN RBA=LRURBA2;
       ELSE IF INDEXLVL=4 THEN RBA=LRURBA3;
    INDEXLVL=INDEXLVL-1;
  M=1;                 ***** LOOK FOR A MATCH IN THE BUFFER;
  DO WHILE (M<= NUMOFBUF AND NOT
                   (AFIL(M)=FILENAME AND ARBA(M)=RBA));
       M=M+1; END;
  IF M>NUMOFBUF THEN DO; M=1; HIT=0;    ***** BUFFER MISS;
                          SIMULIO+1; END;
                 ELSE IF READNEW THEN DO; *** SHAREOPT. 4;
                          HIT=0;SIMULIO+1; END;
                 ELSE HIT=1;                ***** BUFFER HIT;
  DO WHILE (M< NUMOFBUF );  *** PLACE MOST RECENT AT TOP;
       AFIL(M) = AFIL(M+1);
       ARBA(M) = ARBA(M+1);
       M=M+1;
     END;
  AFIL(M) = FILENAME; ARBA(M) = RBA;
END;
PROC APPEND DATA=S&BUFSZ BASE=NUMOFRDS;
RUN;
%MEND;
 %BUFANAL(8192,10,0);   * <== MODIFY, ADD, DELETE THE;
 %BUFANAL(2048,18),0;   * <== %BUFANAL STATEMENTS FOR;
 %BUFANAL(2048,100),0; * <== ENVIRONMENT BEING SIMULATED;
OPTIONS LINESIZE=80;
PROC PRINT;  ID POOLNUM SZOFBUF;
  VAR NUMOFBUF NUMOFREC NUMOFMBY NUMOFIO ACTBYTIO HITR
      SIMULIO SIMBYTIO SIMULHR;
  SUM NUMOFREC NUMOFMBY NUMOFIO ACTBYTIO SIMULIO SIMBYTIO;
```