A SIMULATION STUDY COMPARING FIVE CONSISTENCY ALGORITHMS FOR REDUNDANT DATABASES

by

KEVIN E. NORSWORTHY

B.G.S. University of Kansas, Lawrence, 1977

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

Kansas State University

Manhattan, Kansas

1978

Approved by

Major Professor

TABLE OF CONTENTS

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

LIST OF FIGURES

ACKNOWLEDGEMENTS

I wish to express my gratitude to Drs. Fred Maryanski, Beth Unger, and Ed Basham for serving on my committee and offering their helpful comments and corrections. A special thanks is given to Dr. Maryanski for his help and guidance in the preparation of this report.

KEVIN NORSWORTHY

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

In a distributed database management system it is often desirable to store copies of the same data on different backend machines. There are a number of advantages that can be gained from maintaining duplicate databases. Included among these advantages are increased data accessability, more responsive access, and increased reliability of the data.

Along with these benefits come certain penalties that must be paid for the convenience of having duplicate databases. This report is concerned with the problem of insuring that the duplicate databases are consistent. The design of a system to maintain redundant, duplicate databases is a challenging task because of the inherent communication time delay between copies of the database as well as the real world constraints of system crashes, operator error, communication channel failure, etc.

Several researchers have devised algorithms to insure consistency among replicated databases. To date there has not been a study undertaken to compare the performance of

the different algcrithms. This study was undertaken to accomplish such a comparison by cding five of the algorithms in GPSS-V and comparing their performance as measured by the amount cf time taken to completely process several sample request streams.

1.2 BACKGRCUND

Distributed database management systems have evolved as organizaticns found the need to simultanecusly expand their data prccessing power and increase their data accessability. An econcmical means of satisfying that need can be realized by use of a distributed data base management system comprised primarily of minicomputers. Maryanski et al. (7), Maryanski (9) and Canaday et al. (2) have listed potential benefits that can be realized from using a minicomputer to perform data base functicns. The advantages of a backend processcr are summarized below.

1) It may free the resources of the host machine.

2) It reduces software overhead on the host.

3) It provides additicnal concurrency in the system.

4) It permits additicnal applications to be run cn the host thus increasing through put.

5) It can provide additional security since the priveledge cf executing an applicatiocn prcgram on the

backend can be carefully guarded.

6) It can increase integrity since the host and backend each have the ability to verify the operation of the other.

7) It may be an economic alternative to upgrading the host processor.

Maryanski and Kreimer (8) conducted a simulation study to determine the conditions under which it is feasible, in terms of performance, to distribute the database to configurations of two or three processors. The simulation results indicate that adding additional processors yields performance benefits if the demands for the function are high.

A multiprocessor backend system is quite similar to a distributed data base management system. The design is one of a host coupled to a backend computer with multiple processors. This configuration is shown in Figure 1.1. The advantages of this configuration are that it can reduce communication costs, by elimanating the external channel links between the backends, and may be easier to keep tight security on as access to only one machine must be controlled. A multiprocessor backend should behave like several backend machines with the exception that requests cannot originate at the backend nodes.

Additional benefits can be realized by assigning a copy

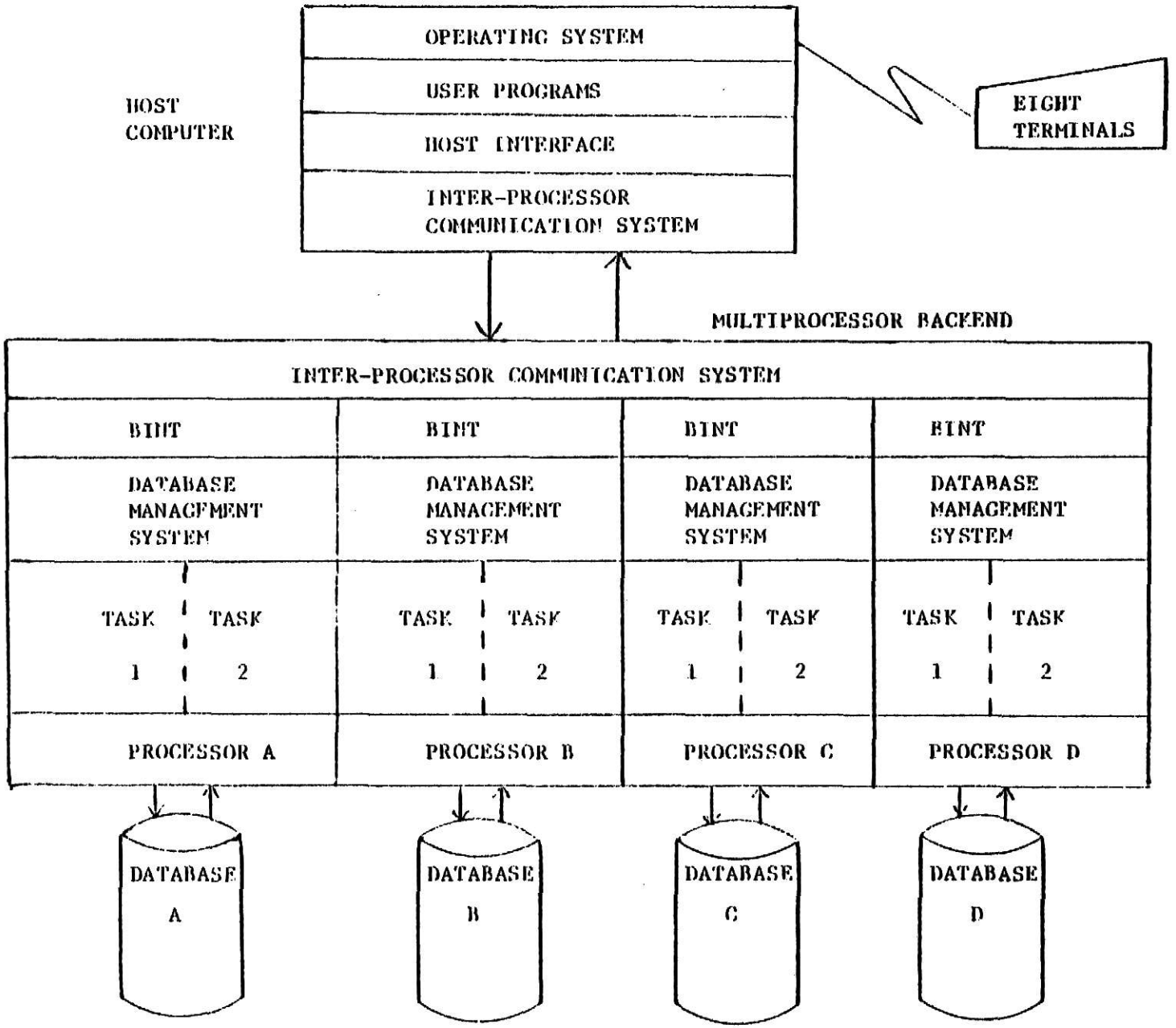

HOST
COMPUTER
OPERATING SYSTEM
USER PROGRAMS
HOST INTERFACE
INTER-PROCESSOR
COMMUNICATION SYSTEM
EIGHT
TERMINALS
MULTIPROCESSOR BACKEND
INTER-PROCESSOR COMMUNICATION SYSTEM
BINT
BINT
BINT
BINT
DATABASE
MANAGEMENT
SYSTEM
DATABASE
MANAGEMENT
SYSTEM
DATABASE
MANAGEMENT
SYSTEM
DATABASE
MANAGEMENT
SYSTEM
TASK
1
TASK
2
TASK
1
TASK
2
TASK
1
TASK
2
TASK
1
TASK
2
PROCESSOR A
PROCESSOR B
PROCESSOR C
PROCESSOR D
DATABASE
A
DATABASE
B
DATABASE
C
DATABASE
D

FIGURE 1.1

of the data base to each backend processor in the system. The multiple copies of the database are referred to as redundant or replicated databases. Johnson and Thomas (6) enumerated several benefits that can be gained from maintaining duplicate databases. These are:

1) Increased data accessability. Data may be accessed even when some of the machines are inoperative.

2) More responsive access. The host processor can access data that is located at the least busy or closest backend.

3) Increased throughput. The computational load can be shared by several machines and

4) Increased reliability of data. Multiple copies assure a reliable backup in case of software/hardware malfunction on one of the machines.

Having outlined the benefits of distributed databases and redundant database processing one can envision a situation where a multiprocessor backend linked to fully replicated databases might best serve the needs of the user. This is the environment that was simulated in this study.

One of the inherent problems with redundant databases is keeping the copies consistent. There is an important distinction between keeping the copies consistent and

keeping the copies identical. Identical is defined as yielding the same result to a query regardless of which backend processor receives the request. Consistent is defined as having the copies identical after all of the modifications have been executed to completion on all of the machines (Johnson and Thomas, 1975). This study compares the performance of five models that seek to maintain consistent databases.

The five models simulated are the Codasyl model (4), the Hybrid model, the Bernstein model (1), the Ellis model (3), and the Johnson and Thomas model (6). Each of these models are explained in detail in Chapter 2.

CHAPTER 2

THE MODELS

2.1 OVERVIEW

This chapter presents the five models used in this study. For all models the environment simulated was one of fully replicated databases attached to backend processors. The number of backend processors attached to the host was either two, four, or eight. Each of the backend machines had two partitions allocated to executing jobs and a buffer to hold queued jobs. The number of terminals attached to the host processor was eight and the speed of the terminal to host processor channel was 50 K Baud. In all simulations, fifteen users were simulated each issuing from one to seventeen requests with the average number of requests being eight. The requests were of two types, read only or modification. Modification requests were defined as being either the update, insertion, or deletion of records. The percentage of modification type requests was set at either 20%, 35%, or 50%. The dependent measurement for these models was the time needed by the system to service all fifteen users and make all of the databases identical.

A few terms need to be defined before detailing the

actions of each model. A timestamp is the tagging of a request with the absolute clock time of the system. A primary backend is the backend selected to receive the modification command. A secondary backend is the backend selected to receive a modification request after it has been completed at a primary backend.

2.2 CODASYL MODEL

The Codasyl model (4) is unique in its simplicity and guarantees one up-to-date database and several day-old databases. Basically, this proposal directs all modification commands to a single database (the primary database) which is maintained up-to-date throughout the day. The modification commands processed by the primary database are logged and issued to all the other database nodes in the network when second shift begins and processing in the network has decreased.

The advantages of the Codasyl model are its simplicity, guarantee of one correct and current database, and no chance that the databases will be inconsistent after all of the modifications saved by the primary database have been completed on the secondary database.

The disadvantages of the proposed Codasyl model are that since all modification commands must be sent to the same primary backend there will be a high likelihood of the

primary backend becoming overloaded. Also, read-only type requests issued during first shift will likely receive day-old information. Finally, there is a buffering problem concerning the receiving of modification commands by the secondary backends.

2.3 HYBRID MODEL.

A model quite similar to the Codasyl model was developed by the author. As with the Codasyl model, all modification type requests are directed to a designated primary backend for processing after which they are logged on the primary backend's modification list. The difference between the Hybrid model and the model proposed by the Codasyl group is the timing of when the modifications are sent to the other backends. As mentioned previously, the Codasyl model issues modification commands to the secondary backends during second shift. The Hybrid model forwards modification commands to the secondary backends whenever the primary backend encounters a lull in processing as signified by an empty backend partition. Timestamping of the modifications was carried out at time of execution by the primary backend so that a user can reference the modification list of the backend he is using to determine the currency of the referenced information.

The advantages of this model are its simplicity, guarantee of one current and correct database, the

elimination of inccnsistency after all cf the modifications saved by the primary database have been completed cn the secondary databases, and the currency of the information stored in the seccndary databases. When processing is light cn the primary database, all of the seccndary databases will contain informaticn that is nearly current. The major disadvantage of this model is the possibility cf overlcading the primary backend with modificaticn requests.

2.4 ELLIS MCDEL

Ellis (3) presented a decentralized approach tc the management of redundant databases. With his model modificaticn requests may originate at any node in the network. The node of origin broadcasts a request to modify the reccrd specified to all other nodes in the system. The node cf origin then waits, blocking out other requests, until it has received positive acknowledgement from all other nodes in the network. If it receives positive acknowledgement from the other nodes, it executes the modificaticn and sends the modificaticn tc the other nodes in the retwcrk. If the node of origin receives a negative acknowledgement frcm any of the other ncdes, it suspends activity cn the modification and resubmits it again at a later date. Pcsitive acknowledgement is given if the seccndary backend is not currently processing a modification

request for the same record.

Positive acknowledgement was a necessary part of the Ellis model because modification requests could originate at any of the database nodes. In this study, all requests were first routed through the host machine so the acknowledgement process was not deemed necessary. Hence, the model was modified as detailed in the following paragraph.

When a modification request was recognized by the host processor, it was simultaneously forwarded to all backends to join their job queues. As all requests were first passed to the host processor, there was no chance of requests being sent out of sequence, thus the databases maintain consistency. This algorithm demands centralized supervision of the processing of requests.

The advantages of this model are that all databases are current and correct, and that it is simple to operate.

The disadvantages of this system are that the backends can lock out all read requests if there is a spurt of modification requests and that it slows down overall processing.

2.5 BERNSTEIN MODEL

Bernstein et al (1) suggested a methodology for synchronizing concurrent transactions in a decentralized distributed database system. Following this model, the host processor assigns the request to a primary backend based

upon its availability. Upon assignment the request is tagged with a timestamp. If the request is a modification, the primary backend locks out further processing while it sends the record number and timestamp to the secondary backends. The secondary backends then 'vote' whether to accept or reject the modification request. Acceptance is granted if no currently executing jobs on the backend reference the record number of the current modification request. If there is another request that conflicts, then the request is delayed and resubmitted later.

When the primary backend has received positive responses from all of the secondary backends, it completes the modification request and forwards it to the other backends for processing.

The advantages of this model are that it guarantees consistency in the databases and provides correct and nearly up-to-date information at all backends.

The disadvantages of this model is that it may require substantial overhead and system time in backend to backend communication during the voting process and it locks out all requests at the primary backend during the entire voting sequence.

2.6 JOHNSON AND THOMAS MODEL

Johnson and Thomas (6) proposed a consistency model for requests made in a decentralized data base environment

incorporating the use of timestamps. With their model requests were timestamped and then assigned to a backend processor based on a backend utilization criteria. After execution of a modification command, the request, with its tagged timestamp, is added to the backends update list which is simply a table of the modifications the backend has processed up to that time. The modification is also marked as a candidate modification that needs to be executed on the secondary backends when the primary backend encounters a lull in processing.

When a backend is free (recognized by an empty partition), it sends its oldest candidate modification to the other backends. At the secondary backend, the request's record number is compared to the update list for that secondary backend for a match. If a match is found and the request on the secondary backend's update list has a more recent timestamp, the modification from the primary backend is ignored. If, after comparison, the modification from the primary backend is found to be the most recent for that record, it is processed and added to the secondary backend's update list. A third case also exists. It may be that the modification from the primary backend is more recent than a modification in the secondary backend's job queue. In this case the modification in the job queue is deleted in favor of the more recent modification. This third case can cause inconsistency between the duplicate databases.

The advantages of this model are that it attempts to

keep all databases current by amending them throughout the day. Also, the timestamp prevents modification of a record by a modification older than the most recent modification of that record at a particular backend.

A disadvantage of this model is that it allows for inconsistency among the databases.

CHAPTER 3

SIMULATION IMPLEMENTATION

3.1 OVERVIEW

Our simulation implementation of the models along with general assumptions and parameters used will be discussed in this section. Additionally, the Bernstein model will be described in detail to aid the readers comprehension of the code. Following the explanation of the Bernstein model will be a detailing of the modifications necessary to produce the code for the other four models.

3.2 MODEL DESCRIPTION

Appendix 1 contains the listings of the five simulation models. The systems we modeled in GPSS-V and run on an ITEL AS5. The programs required approximately 30 seconds CPU time for each simulation, although this figure varied with the model and program parameters. The simulation runtime varied between 40 seconds and two minutes. This was felt to be long enough to obtain good random distributions and a reasonable measurement of throughput. The basic time unit for the simulations was one millisecond.

Each model was run varying the parameters twelve ways.

For each parameter setting, six runs were obtained using different random number seeds obtained from the GPSS V User's Manual (5). Occasionally, only five runs could be obtained because one of the six runs would exceed GPSS V limits on common core. For three of the five models, GPSS System Option C was needed which requires 178 K for storage. For the other two models the 104 K provided by GPSS System Option E were sufficient.

The twelve different parameter settings were of two types. The following nine were run using a 50 K Baud line between the host and backend and between the backends themselves. The environments were:

1) Two backends with 20%, 35%, and 50% modification requests.

2) Four backends with 20%, 35%, and 50% modification requests.

3) Eight backends with 20%, 35%, and 50% modification requests.

Three environments were simulated with no time delay between the host and backends or from the channel connections between the backends. This environment represented the architecture of a single host machine with attached processors (the backends) and extended memory. The three system environments simulated for each model using

this architecture were:

1) Two backends receiving 50% modification requests,

2) Four backends receiving 50% modification requests,

3) And eight backends receiving 50% modification requests.

Each of the five models was run in all twelve environments six times. The random number generator seeds were changed to produce different job mixes.

The code of all five of the models was similar. Therefore, only the Bernstein model, the most complex model, will be explained in detail. Following this explanation will be a detailing of the modifications of the Bernstein model that were required to produce the other four models.

3.3 EXPLANATION OF THE BERNSTEIN MODEL

During the simulation, fifteen users sign on to the system. The users sign on at approximately 1.5 second intervals. The users are randomly assigned to one of the eight terminals attached to the host processor. Each user issues 1 to 17 requests with the average being 8 requests.

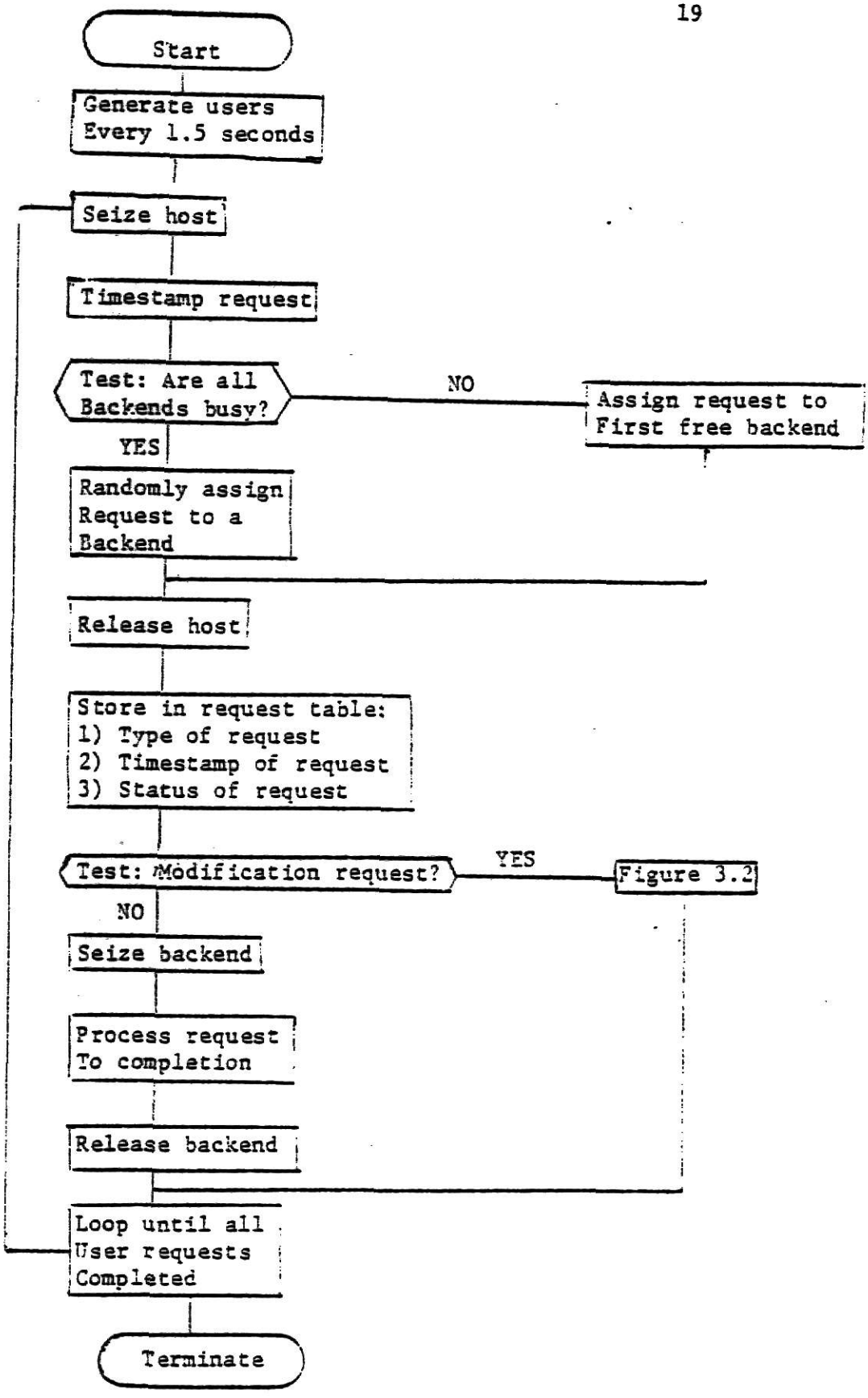
Transmission to the host from the terminals is by a 50 K Baud channel. Ninety percent of the requests fit in one

buffered transmission while the other ten percent require two buffered transmissions. The data for these parameters was supplied by researchers at Kansas State University based upon measurements of their distributed database system.

After the user has typed in his request the message is transmitted to the host where it is tagged with the record number and the absolute clock time (a timestamp). Also, the number of I/O operations required by the request is assigned. The number of I/O operations ranges from a minimum of 0 to a maximum of 14 with a mean of 8. The host then searches each of the backends until it finds one that is free. It accomplishes this by sending messages across the host to backend channels which run at 50 K Baud or infinite speed. It assigns the request to the first free machine or randomly selects a backend if all of them are busy.

At the chosen backend (hereafter referred to as the primary backend) BINT, the backend interface, unpacks the message. It then stores the request, its type, its timestamp and a mark that indicates it is on the backend's request list pending execution. If the request is a read only command, it is executed by the backend relinquishing the processor during I/O. When the command has completed executing, the entry for that command in the request list is marked done. The steps executed up to this point are represented by the flowchart in Figure 3.1. If the request is a modification request, the steps presented in the next

FIGURE 3.1



Main Loop of the Bernstein Model

section are executed.

3.4 ACTIONS TAKEN IN PROCESSING OF MODIFICATION REQUESTS

If the request is a modification, it seizes the primary backend's CPU and prevents processing of any other requests by that backend. The primary backend then sends a request to each of the other backends (secondary backends) in the network asking for permission to modify the specified record. This request is given priority over the execution of any requests at the secondary backends. Each secondary backend compares the request and its timestamp to those requests listed on its request list. If the current modification is the most recent request using that record, then permission to modify that record is granted by the backend. If there is an older request that accesses that record which has not been fully processed by the secondary backend, permission for the current modification is delayed. Permission will be granted after all the older requests have been processed.

When permission to modify has been granted by all of the secondary backends, the modification is executed by the primary backend. After execution, it is marked done and the modification is sent to join the job queue at each of the other backends in the network. When processing of the secondary modifications is completed, the request is marked done on the request list and the transaction is terminated.

These actions are flowcharted in Figure 3.2.

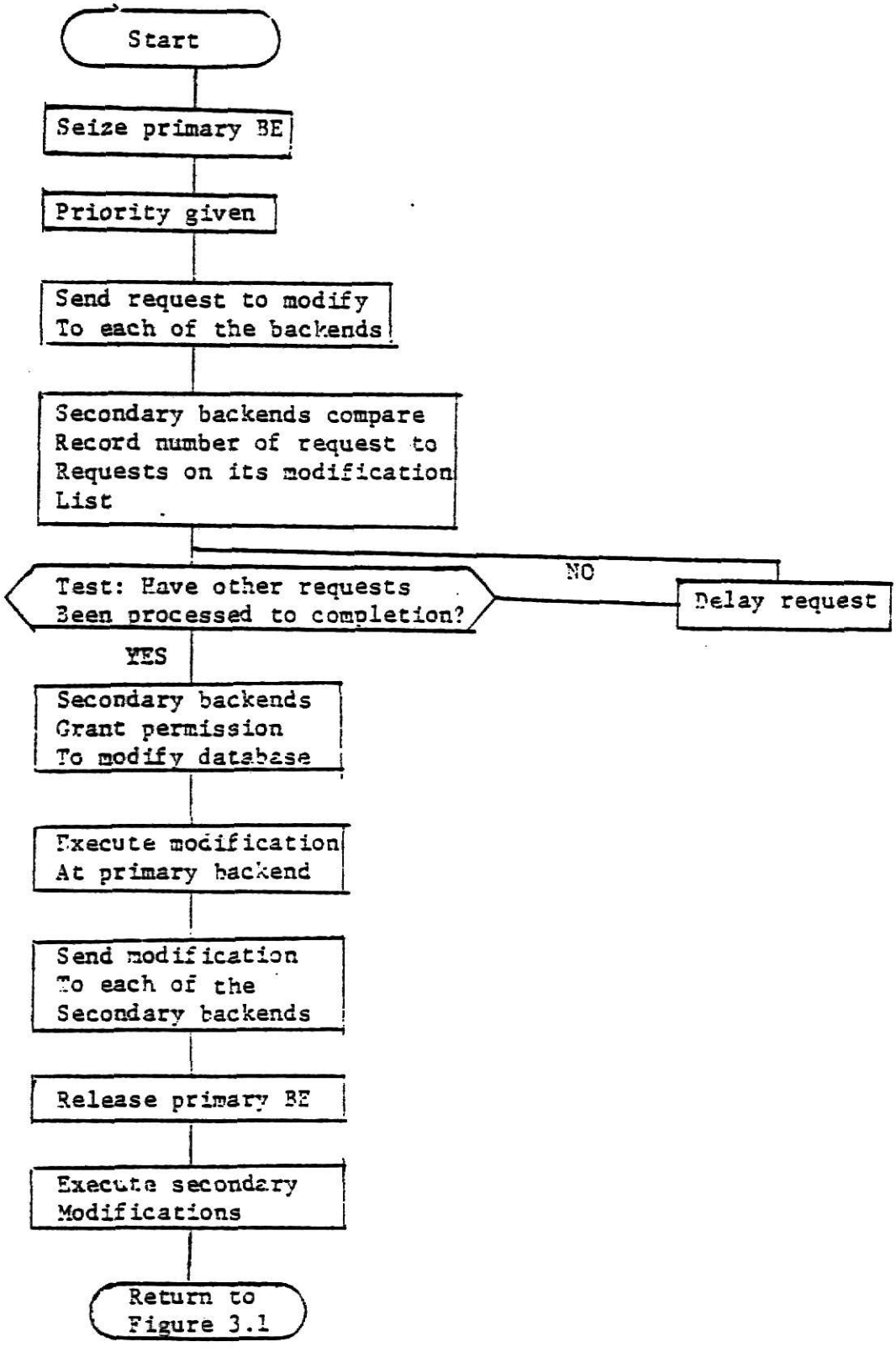
3.5 MODIFICATIONS NECESSARY FOR THE OTHER MODELS

This section will detail the changes necessary to program the four other models. Accompanying each description will be a flowchart of the actions performed.

Several changes were necessary to program the Johnson and Thomas (6) model as detailed in Figure 3.3. First, timestamping was only done if the request was a modification. Also, there is no voting done for this model. Requests are simply processed by the primary backend and placed on the modification list (similar to the request list of the Bernstein model only reserved for modification requests). When the primary backend has a lull in processing it sends the oldest candidate modification on the modification list to the secondary backends where it is compared to the secondary backends modification list. Instead of voting, the backends either accept or reject the modification based upon its timestamp compared to the timestamps of the other modifications on the secondary backends modification list.

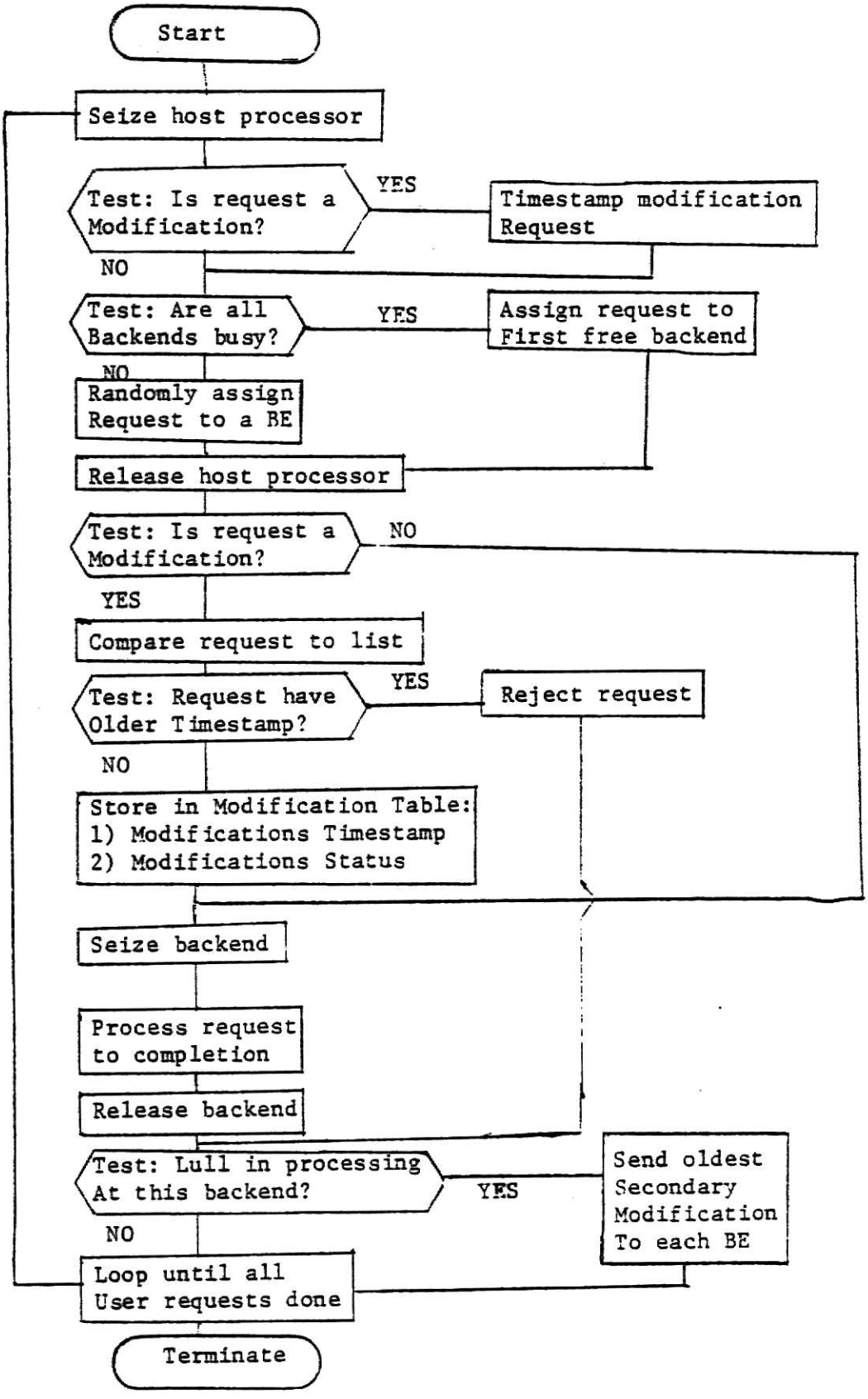
The changes necessary for the Ellis model (3) are detailed in Figure 3.4. Assignment to a backend is the same for read only requests but differs for modification requests. Modification requests are duplicated at the host

FIGURE 3.2



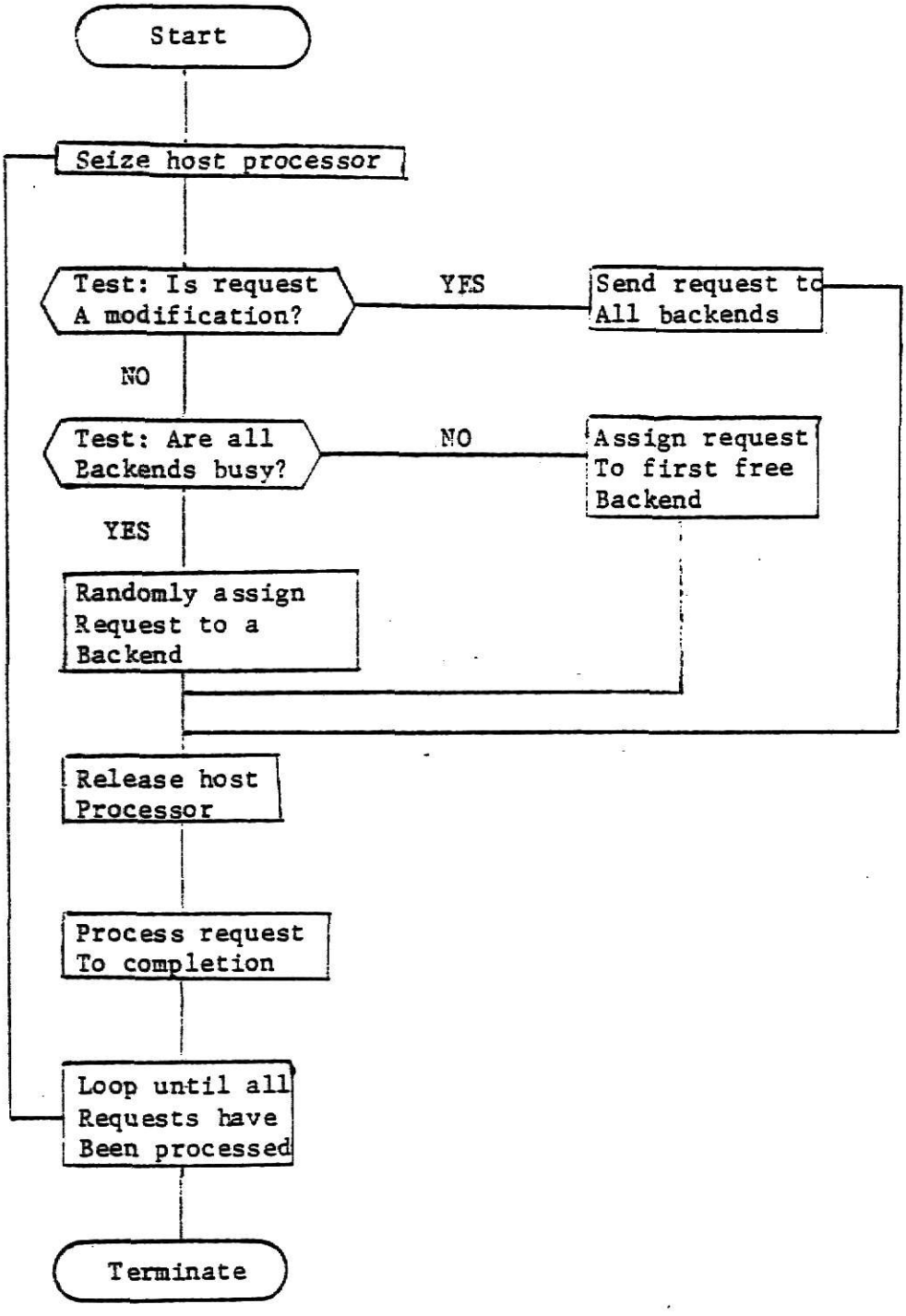
Modification Processing
For The Bernstein Model

FIGURE 3.3



Johnson & Thomas
Model

FIGURE 3.4



Ellis Model

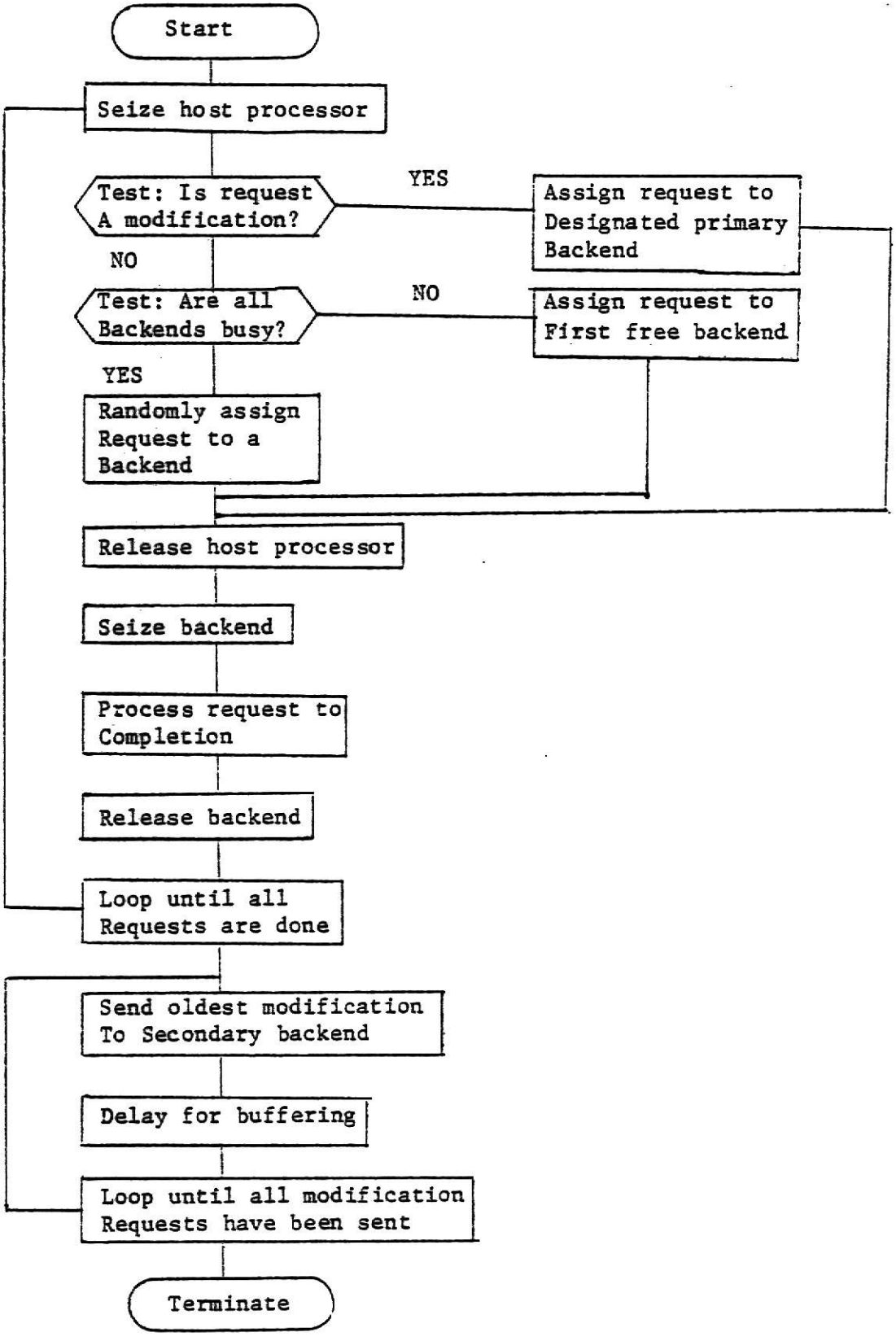
enough times to send a copy to each of the backends in the network. At the backends, the request joins the job queue for the backend and is processed to completion. When the modification has been completed on any of the backends, the user is notified and given the opportunity to enter a new request.

Figure 3.5 details the modifications made to implement the Codasyl model (4). Read only request were assigned to a backend in the same manner as with Bernstein's model. Modification requests were handled differently in that they were assigned to a specified primary backend. No voting was necessary because modifications were simply spooled and then reissued, oldest to newest, after all of the user requests had been processed.

The Hybrid model was almost identical to the Codasyl model. The changes made from Bernstein's model are shown in Figure 3.6. The only modification from the Codasyl model was that after a request has been processed by a backend, the modification list is checked for a pending secondary modification. If there is a pending secondary modification and if there is a lull in processing at the primary backend, then the modification is forwarded to each of the backends in the network.

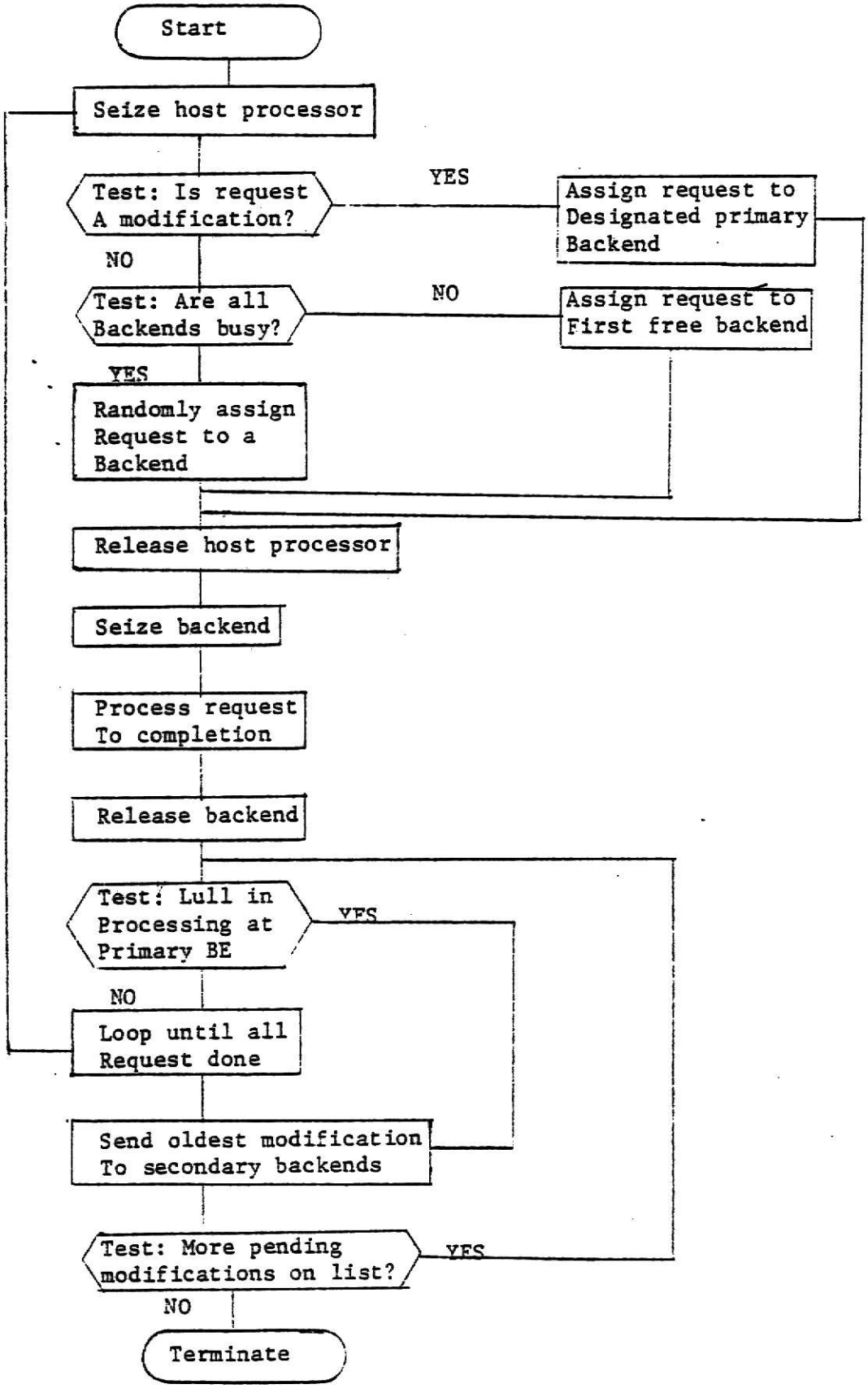
Hybrid II was a model that was tested to determine if the performance of the Hybrid model could be improved by assigning modification requests to the backend responsible for a particular granule of the database. For example, if

FIGURE 3.5



Codasyl Model

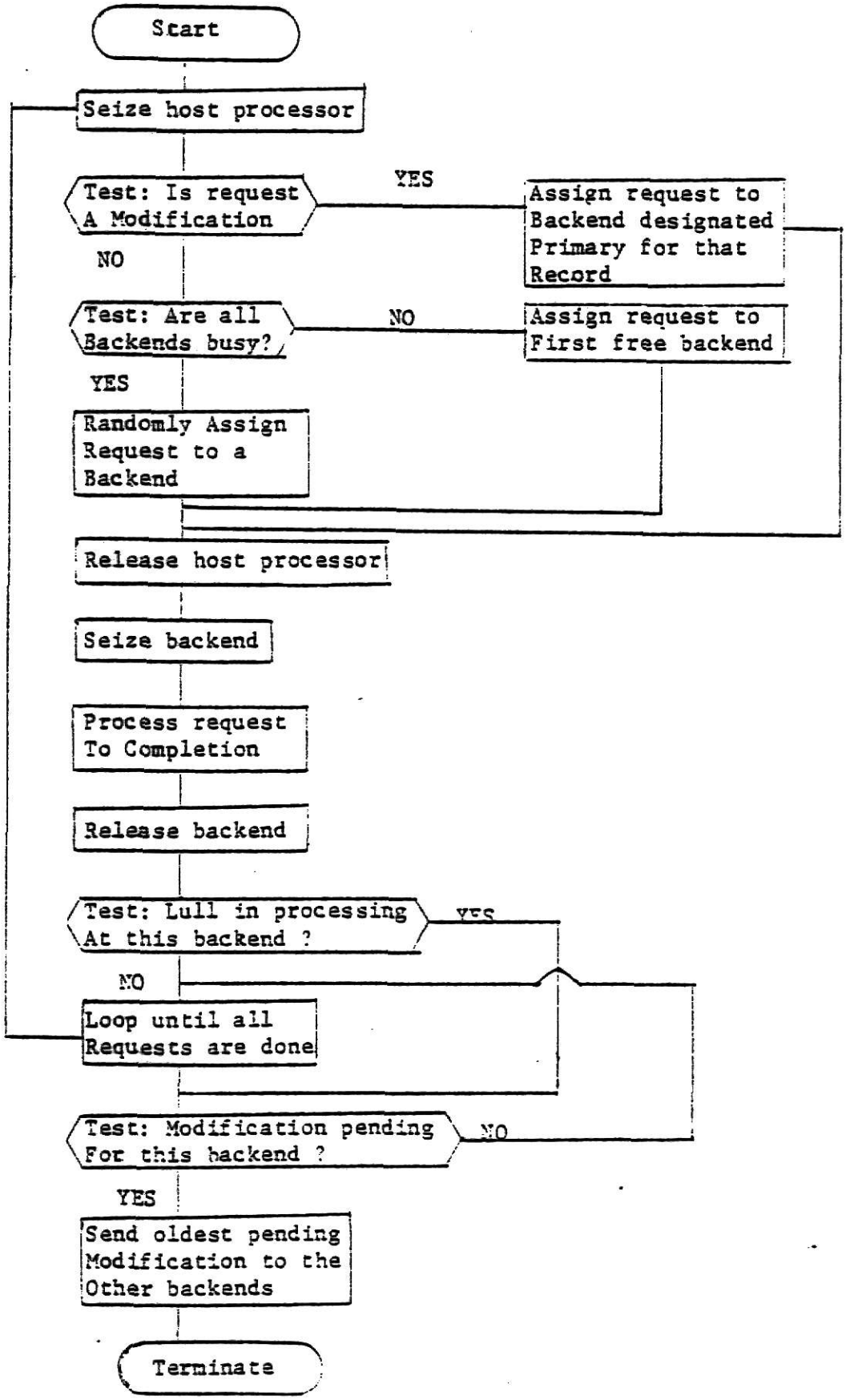
FIGURE 3.6



Hybrid Model

the configuration employed four backends, the database would be divided into four granules with one granule assigned to each of the backends. This would make each backend a primary database for one granule. The modifications from the original Hybrid model are minor and detailed in Figure 3.7.

FIGURE 3.7



Hybrid II Model

CHAPTER 4

RESULTS AND ANALYSIS

4.1 OVERVIEW

This chapter presents the results obtained and an explanation of what might have caused these results. The effect of the model, number of backends employed and percent of modification requests was measured by the amount of system time needed to completely process several sample request streams and make the databases identical. Section 4.2 will examine the global trends of the models and the remaining sections in this chapter will analyze the performance differences of the models.

4.2 GLOBAL TRENDS

A factorial analysis of variance concluded:

1) There was a significant affect ($p < .001$) on performance due to the choice of model

2) There was a significant affect ($p < .025$) on performance due to the number of backends employed in the network

3) There was a significant affect ($p < .001$) on performance due to the percent of modification requests

processed.

The mean performance times for the percent of modification requests processed is presented in Table 1.

PERFORMANCE MEANS FOR THE DIFFERENT
PER CENT MODIFICATION COMMANDS

20% Modification	67127
35% Modification	68518
50% Modification	78739

TABLE 1

The performance means are the time in milliseconds needed to completely process the job stream. Table 1 shows a direct relationship between longer processing time and an increase in the percent of modification requests. This is consistent with the author's a priori hypothesis that modifications lock out other processing causing a potential bottleneck in performance.

The mean performance times for the number of backends used in the network configurations is presented in Table 2.

PERFORMANCE MEANS FOR THE DIFFERENT BACKEND CONFIGURATIONS

2 Backends	74055
4 Backends	70448
8 Backends	69880

TABLE 2

The results show that additional backends increase throughput. They also show that the greatest improvement came with the increase from two to four backends and that only minor improvement resulted from the change from four to eight backends. This finding concurs with the results of a simulation study done by Maryanski and Kreimer (8) comparing performance factors with the addition of extra backends.

The mean performance times for the different models is presented in Table 3.

PERFORMANCE MEANS FOR THE FIVE MODELED ALGORITHMS

Johnson & Thomas	64794
Bernstein	66907
Hybrid	68225
Ellis	73261
Codasyl	84119

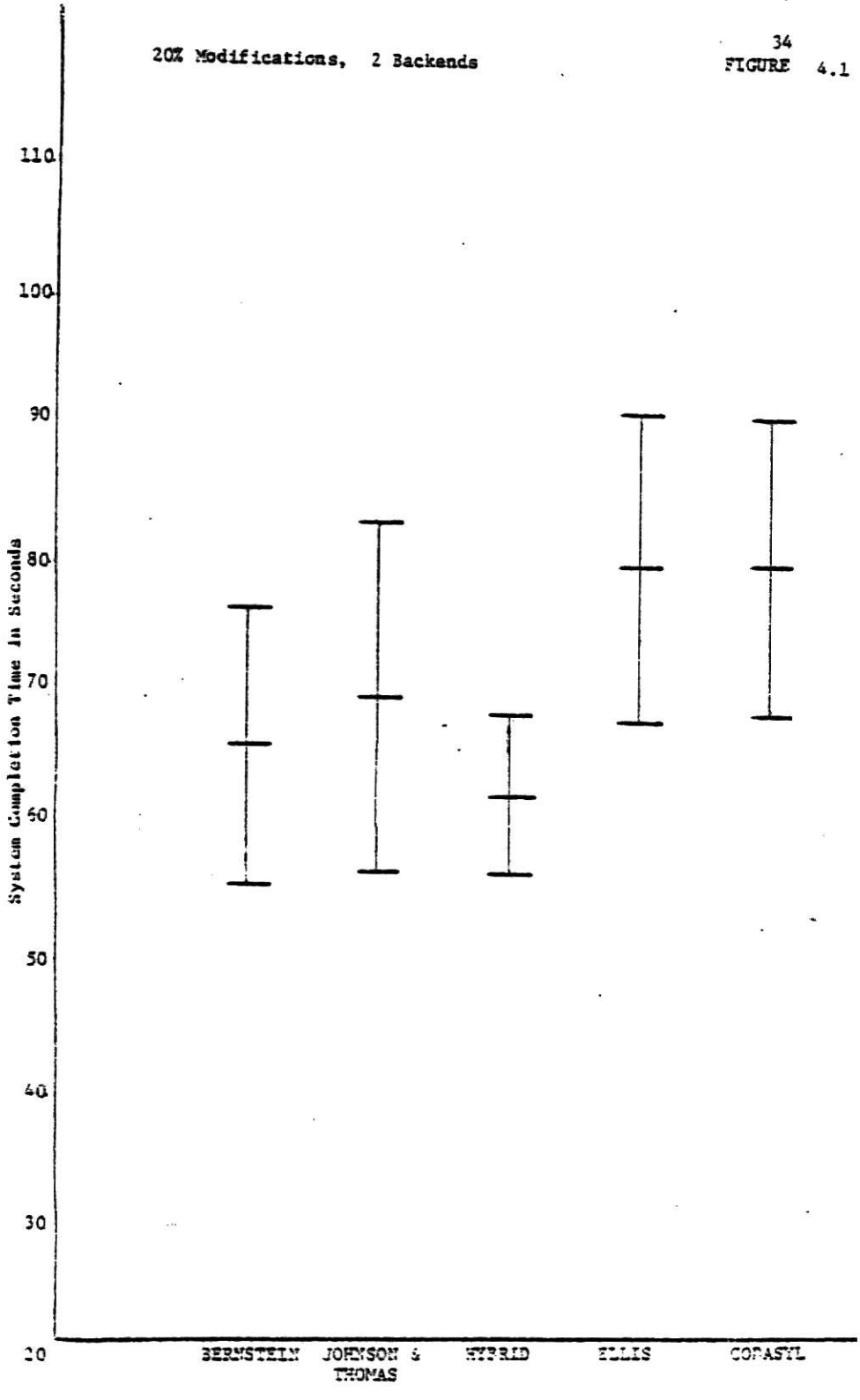
TABLE 3

The results show the Johnson and Thomas model, the Bernstein model and the Hybrid model all processed the sample request stream at approximately the same rate while the Ellis and Codasyl model were slower. A T test was performed to determine which means differed significantly from each other. The results of this test found the differences between the three fastest models was not significant but that the three fastest models differed significantly from the Ellis model ($p < .02$, $p < .01$, $p < .001$) and also differed significantly from the Codasyl model ($p < .001$ for all three). The T test also found that the mean for the Codasyl model differed significantly from the mean for the Ellis model ($p < .001$).

Figure 4.1 shows the performance means and standard deviations of the five models (Bernstein, Johnson and

20% Modifications, 2 Backends

FIGURE 4.1



Thomas, Hybrid, Ellis and Codasyl) in the system configuration of two backends with a job mix of 20% modification requests. In this configuration the Hybrid model performs best with the lowest standard deviation, however, its performance is not significantly different from the Bernstein or Johnson and Thomas model.

Figure 4.2 shows the performance of the models in the same two backend configuration with a job mix of 35% modification requests. In this environment the order of the performance times is the same as the overall mean averages obtained. More confidence can possibly be placed in these results because the standard deviation of all the results is slight.

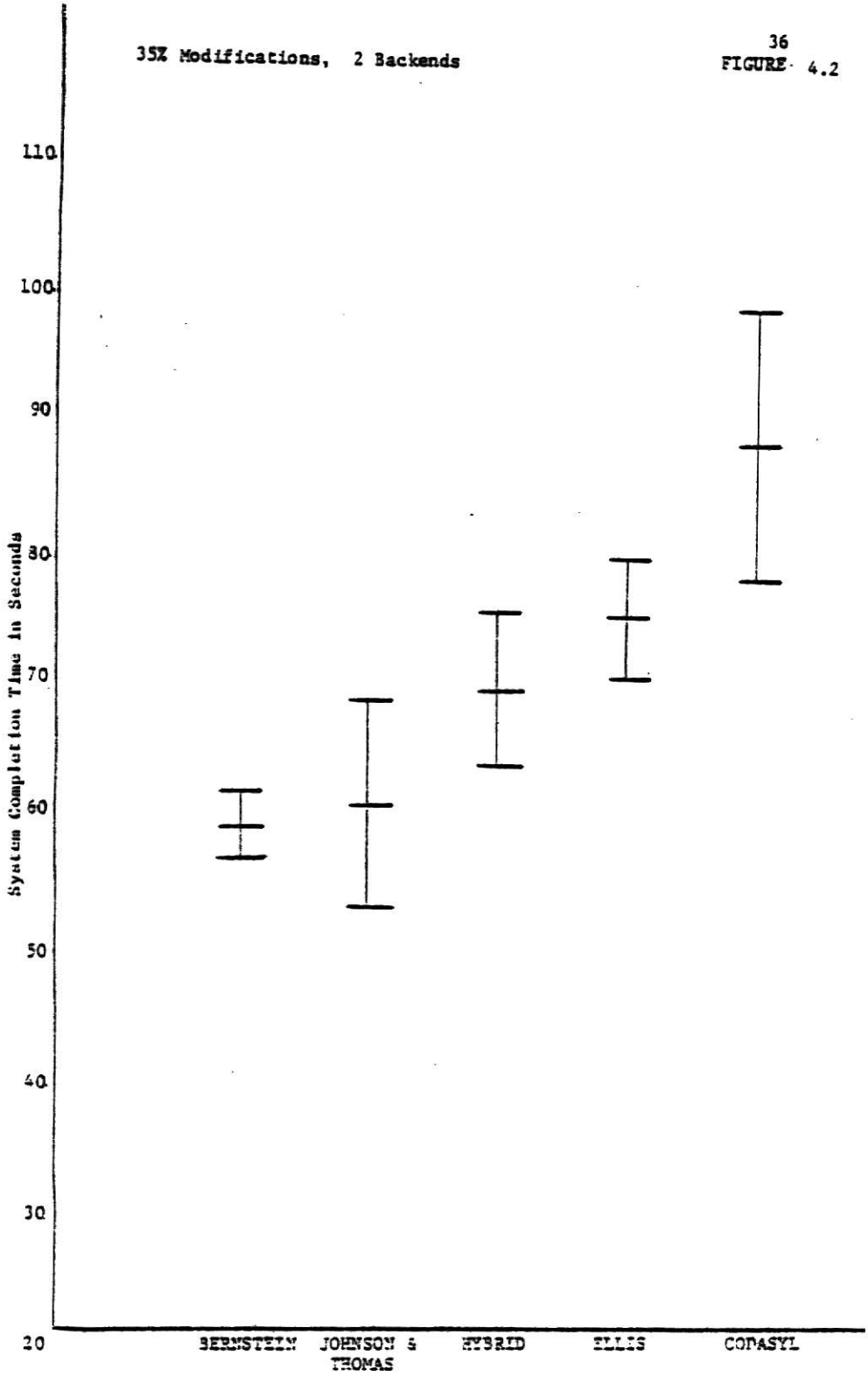
The 50% modification job mix for the same configuration produced results with dramatic standard deviations. This is shown in Figure 4.3. Due to the nearness of the means and the size of the standard deviations, no conclusions can be drawn from these results.

The four backend configuration produce an overall decline in the performance time for all models and all job mixes. For the 20% modification job mix, the Bernstein and the Johnson and Thomas model performed best although the advantage was not great. This is shown in Figure 4.4.

At a 35% modification job mix, the Johnson and Thomas model stands out as the favorite. It's mean performance time was approximately a full second faster than the next closest model and it standard deviation was small. The

35% Modifications, 2 Backends

FIGURE 4.2



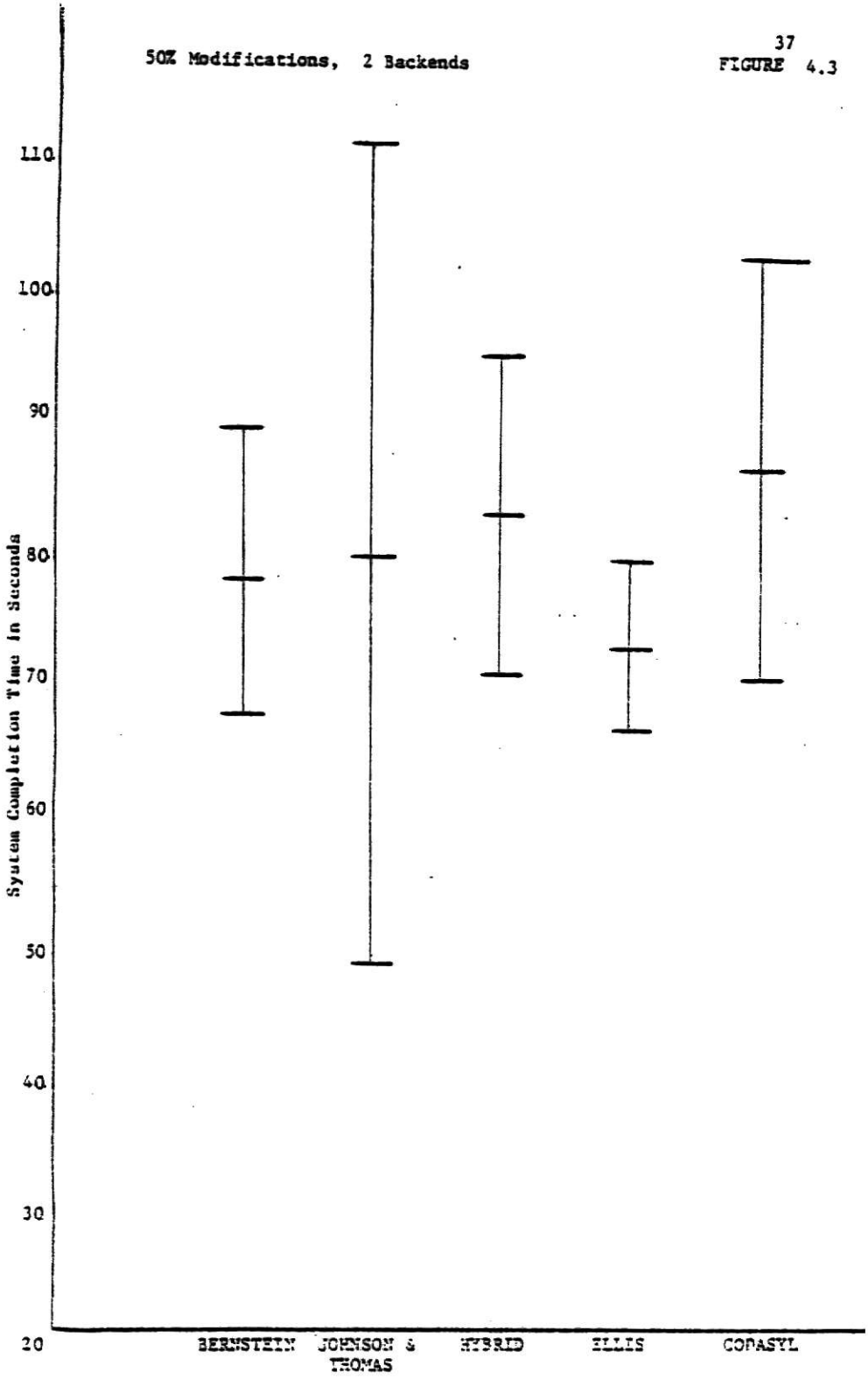
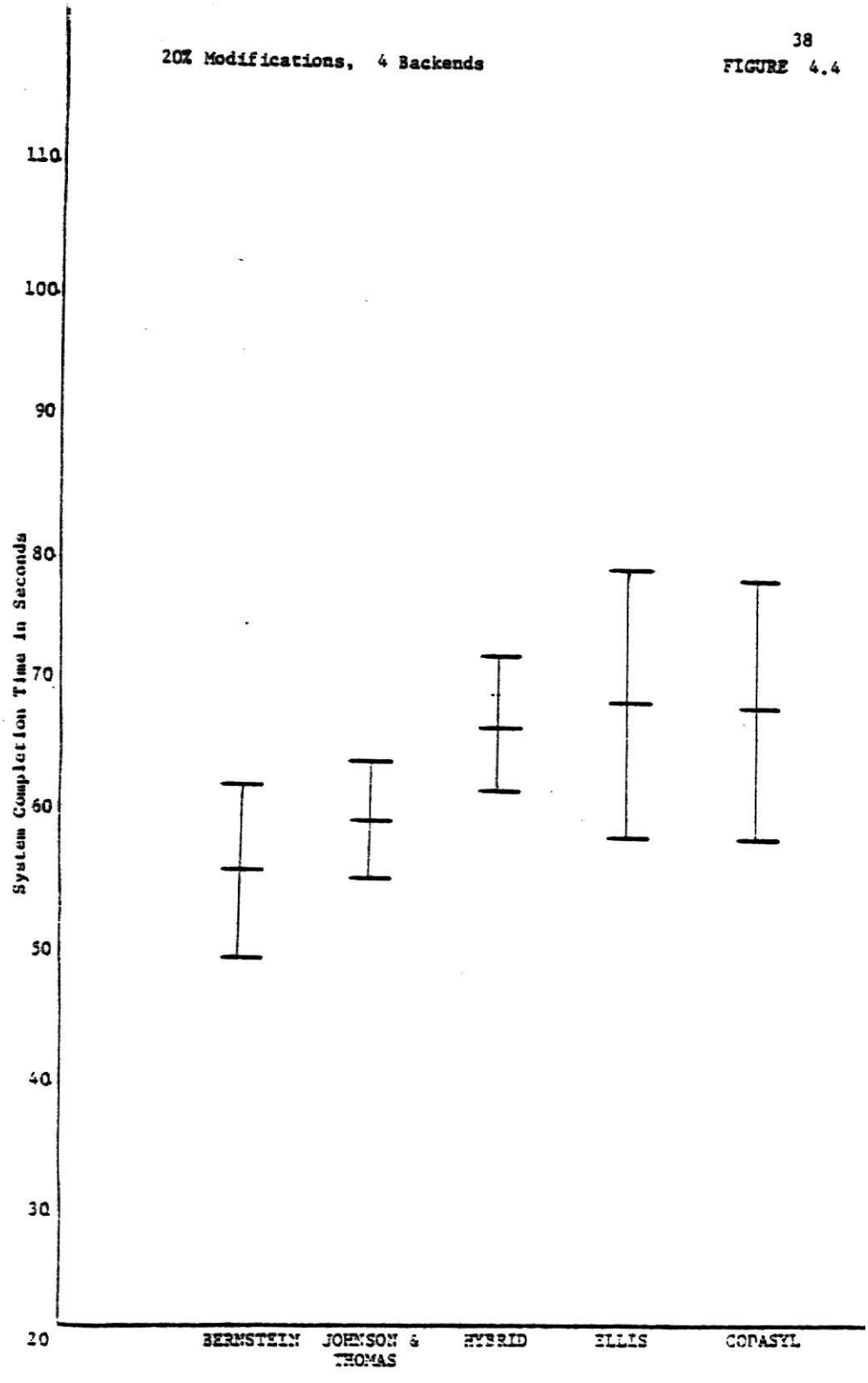

50% Modifications, 2 Backends
System Completion Time in Seconds
110
100
90
80
70
60
50
40
30
20
BERNSTEIN
JOHNSON & THOMAS
HYBRID
ELLIS
COPASYL

FIGURE 4.3

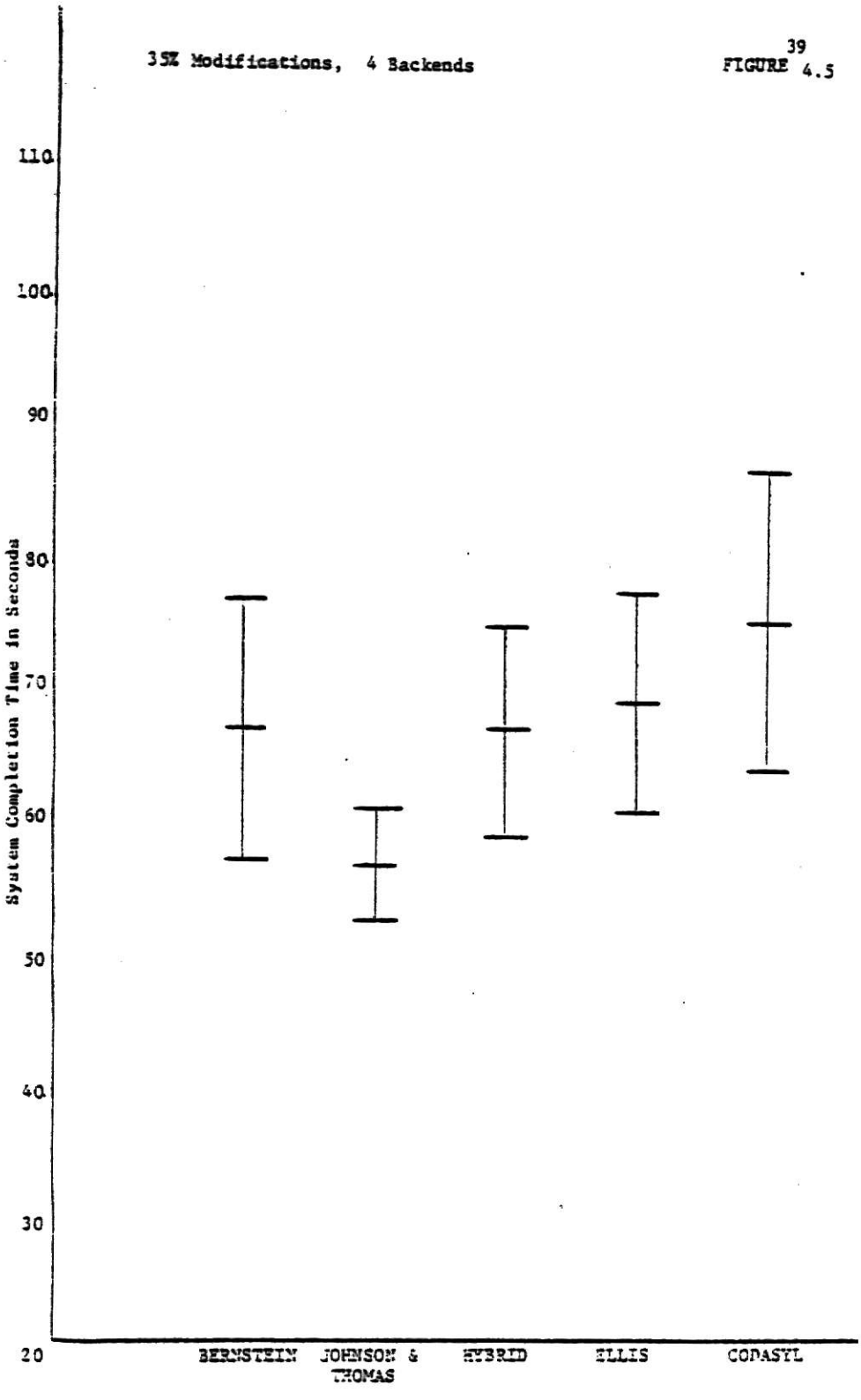
20% Modifications, 4 Backends

FIGURE 4.4



35% Modifications, 4 Backends

FIGURE 4.5



graph for this environment is presented in Figure 4.5.

With a job mix of 50% modification requests in the four backend configuration, the three fastest models performed almost equally. With 50% modification requests a large standard deviation is noted. Figure 4.6 shows the results of runs made in this environment.

As noted before, the advantage of eight backends to four backends was slight. The next three analyses will be of the results obtained for the eight backend environments at the different job mixes.

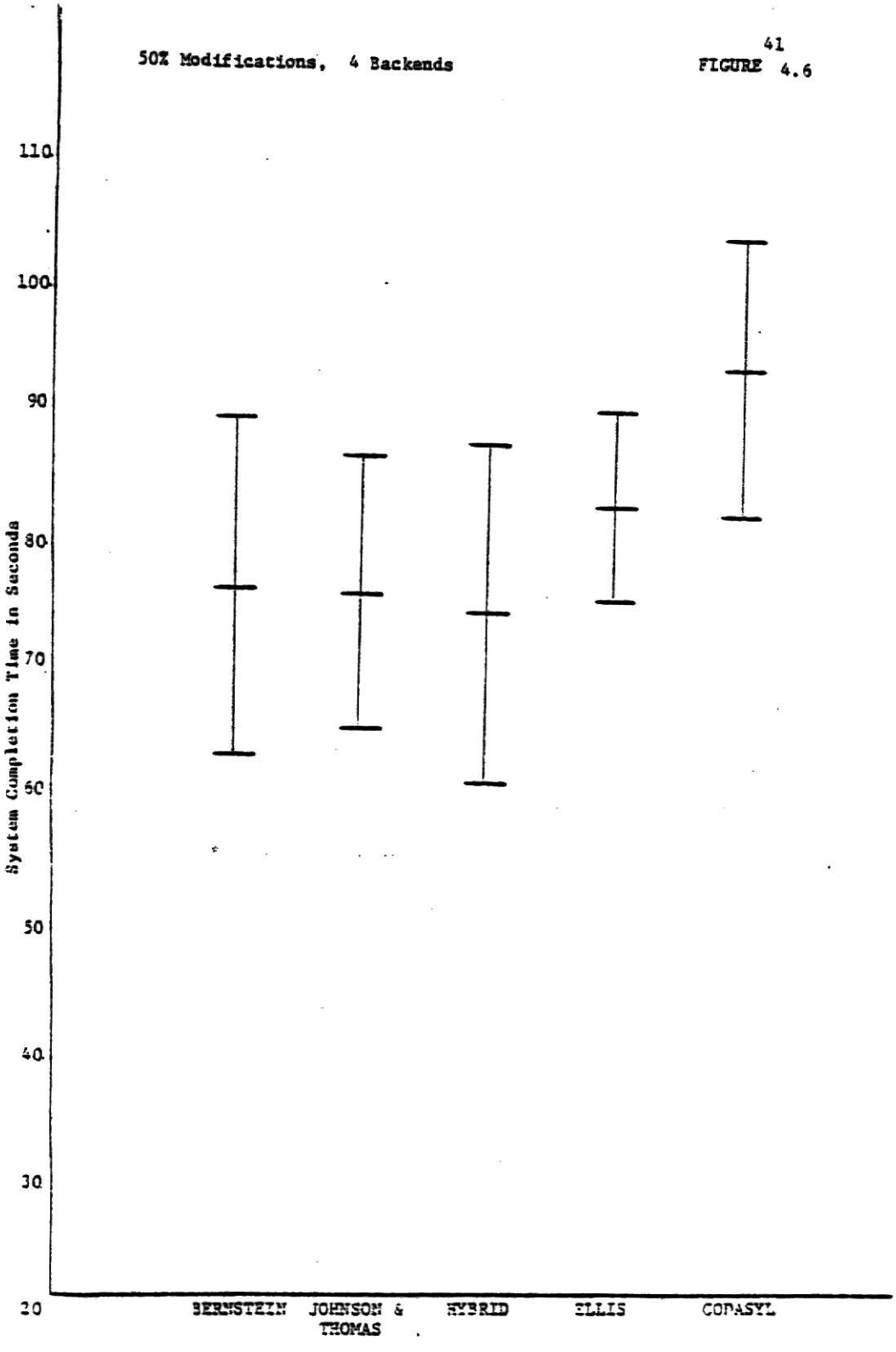
With the 20% modification job mix the three fastest models performed about the same. This graph is shown in Figure 4.7. An interesting result obtained from this graph is that the standard deviation of the Johnson and Thomas model is substantially less than the standard deviation of the Bernstein and Hybrid model.

Figure 4.8 shows the eight backend configuration with a 35% modification job mix. In this environment, all of the models obtained approximately the same mean results with the exception of the Codasyl model which was substantially higher.

The 50% modification job mix showed the greatest variety of results as detailed in Figure 4.9. In this environment the Johnson and Thomas model performed exceedingly well at a low variance. A surprising result was the relatively poor performance of the Bernstein model. Since the variance of the results of the Bernstein model

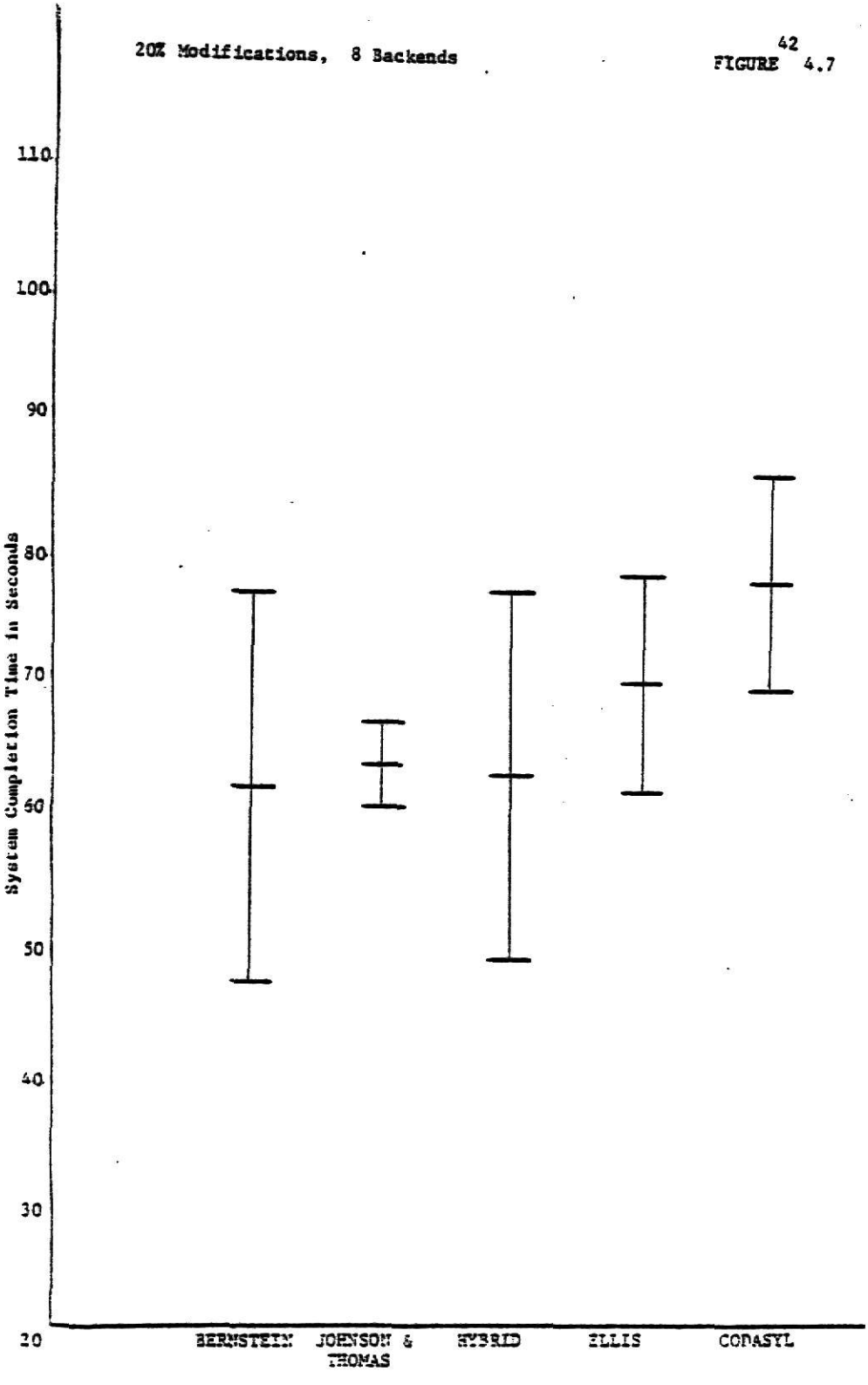
50% Modifications, 4 Backends

FIGURE 4.6



20% Modifications, 8 Backends

FIGURE 4.7



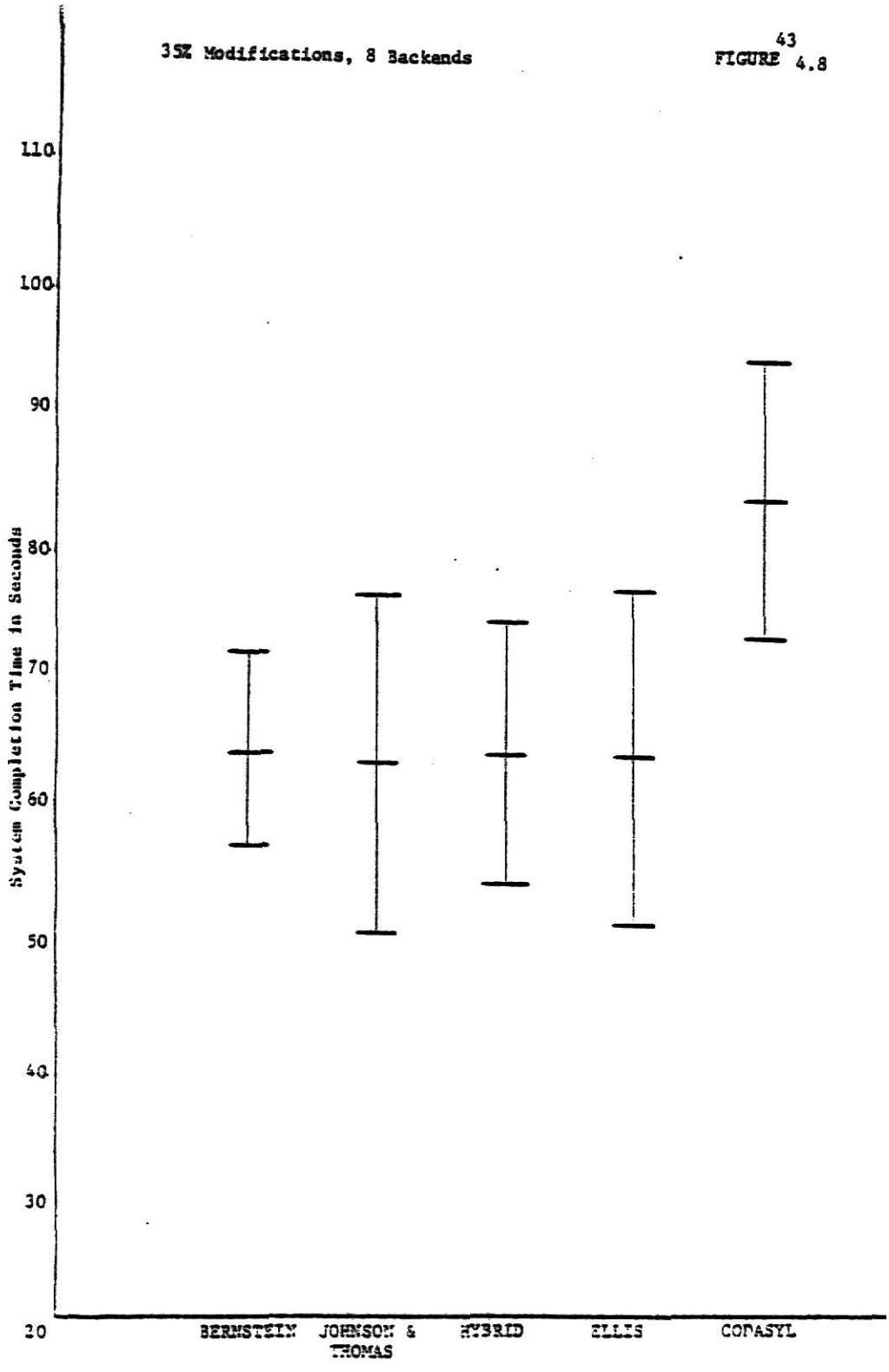
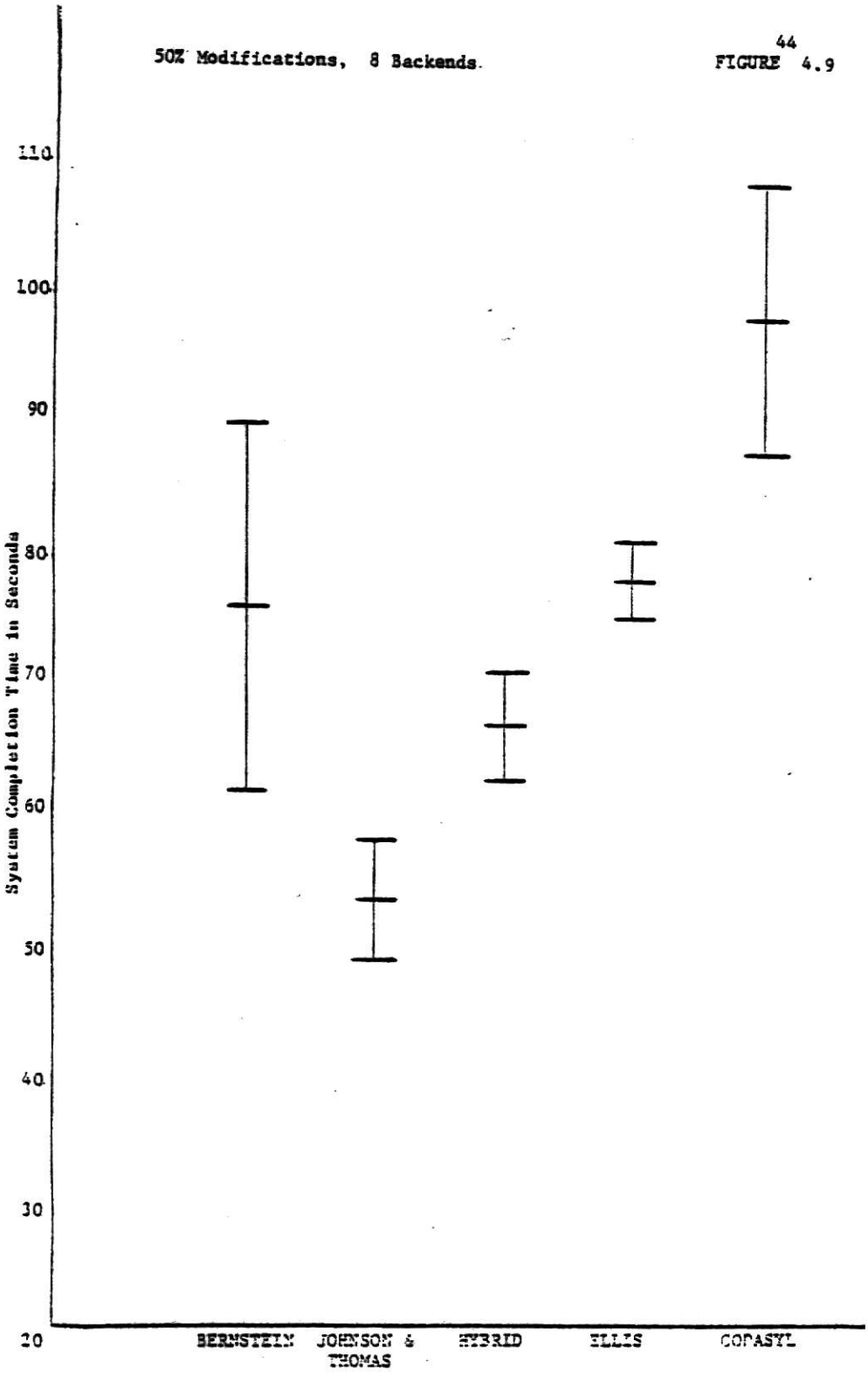

35% Modifications, 8 Backends
System Completion Time in Seconds
110
100
90
80
70
60
50
40
30
20
BERNSTEIN
JOHNSON & THOMAS
HYBRID
ELLIS
CODASYL

FIGURE 4.8

50% Modifications, 8 Backends.

FIGURE 4.9



simulaticns is quite high, the validity of these results is suspect.

Lccking at the graphs as a whole, the figures ccnfirm the statistical finding that the Bernstein, Johnson and Thomas and Hybrid model perform the best, followed by the Ellis model and lastly the Codasyl mcdel. The exact numerical values are presented in Tables 4, 5, and 6 where each table details all the mcdels results for the three job mixes fcr a particular backend configuratien.

One more general trend was noted. Figures 4.10, 4.11, and 4.12 summarize a comparison of the performance times of the models run with 50 K Baud channel speed to identical models run with infinite channel speed. The graphs show a consistent trend that the models prccess the job stream faster with inifinite channel speed. Although the trend was strcng, it failed to achieve significance as measured by a T test.

In summary, the analysis of general trends suggest the follcwing:

1) Increasing the percent cf mcdification commands will slcw system processing.

2) Up to a pcint, increasing the number cf backends in the netwcrk will speed processing of the jcb stream.

3) Three of the models, Bernstein, Hybrid, and Jchnson and Ihcmas, process the jcb stream significantly faster than the other two models.

TABLE 4*

	BERNSTEIN	JOHNSON & THOMAS	HYBRID	ELLIS	CODASYL
20%					
MEAN	64597	68574	61346	78942	78990
ST. DEV.	10795	13465	6052	11750	11045
35%					
MEAN	58997	60785	69179	74978	88520
ST. DEV.	2930	.8695	5656	4724	10668
50%					
MEAN	77715	79509	82648	72547	86066
ST. DEV.	11543	31039	12413	6278	16022

* NOTE: Time is given in milliseconds.

TABLE 5*

	BERNSTEIN	JOHNSON & THOMAS	HYBRID	ELLIS	CODASYL
20%					
MEAN	55784	59578	66908	68969	68245
ST. DEV.	6538	4506	4596	10459	10025
35%					
MEAN	66977	56694	66779	69110	75141
ST. DEV.	10023	4162	7943	8087	11525
50%					
MEAN	75912	75864	74143	82234	93641
ST. DEV.	13169	11206	13795	7470	10125

* NOTE: Time is given in milliseconds

TABLE 6*

	BERNSTEIN	JOHNSON & THOMAS	HYBRID	ELLIS	CODASYL
20%					
MEAN	61482	63818	62362	69882	77068
ST. DEV.	15513	2946	14732	8320	8781
35%					
MEAN	64338	63775	64166	64039	84298
ST. DEV.	7456	12802	10481	12603	10713
50%					
MEAN	76001	53809	66509	78655	98029
ST. DEV.	14794	4535	4104	3526	10580

* NOTE: Time is given in milliseconds

FIGURE 4.10

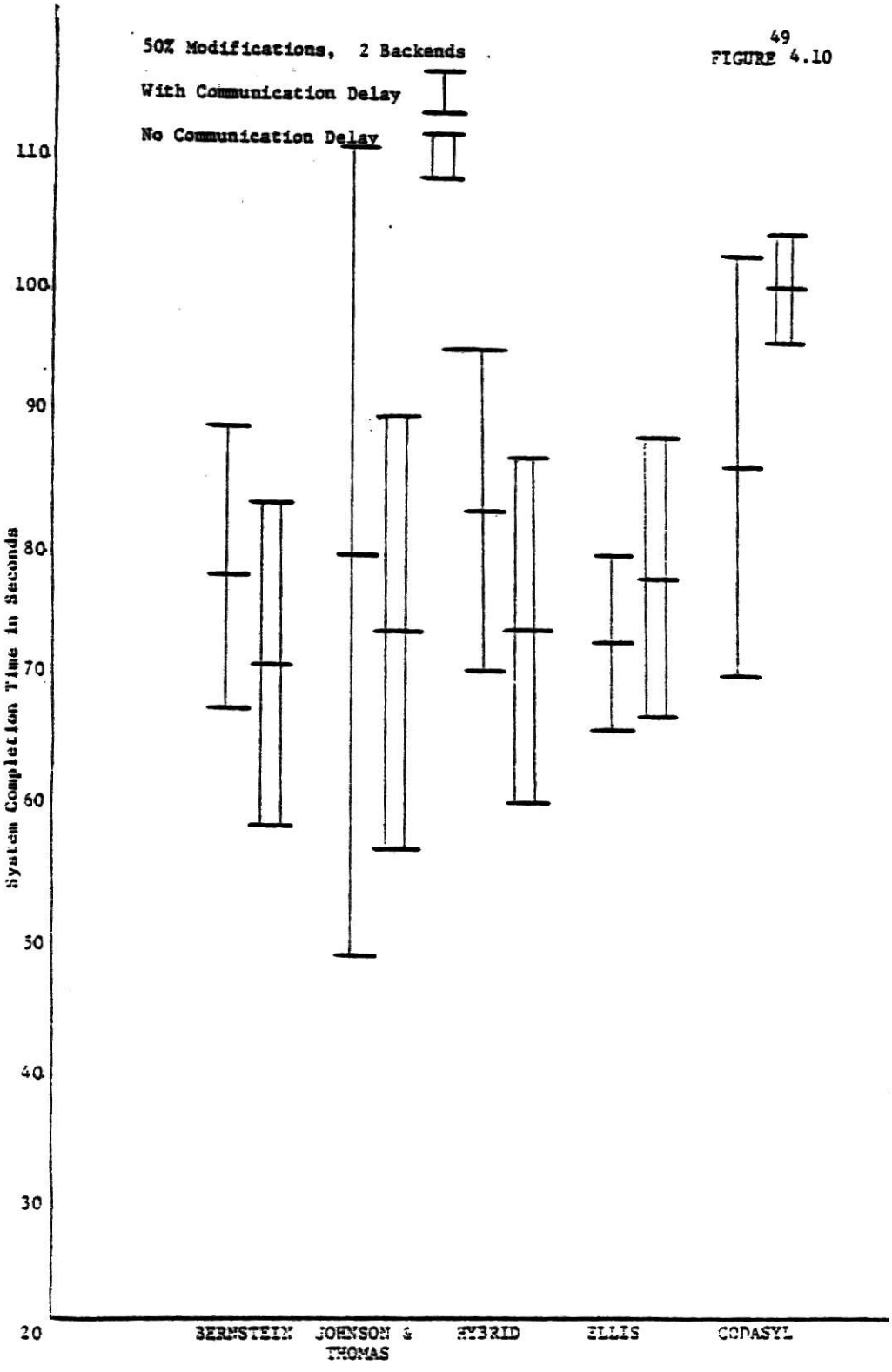

50% Modifications, 2 Backends
With Communication Delay
No Communication Delay
System Completion Time in Seconds
110
100
90
80
70
60
50
40
30
20
BERNSTEIN
JOHNSON & THOMAS
HYBRID
ELLIS
CODASYL

FIGURE 4.11

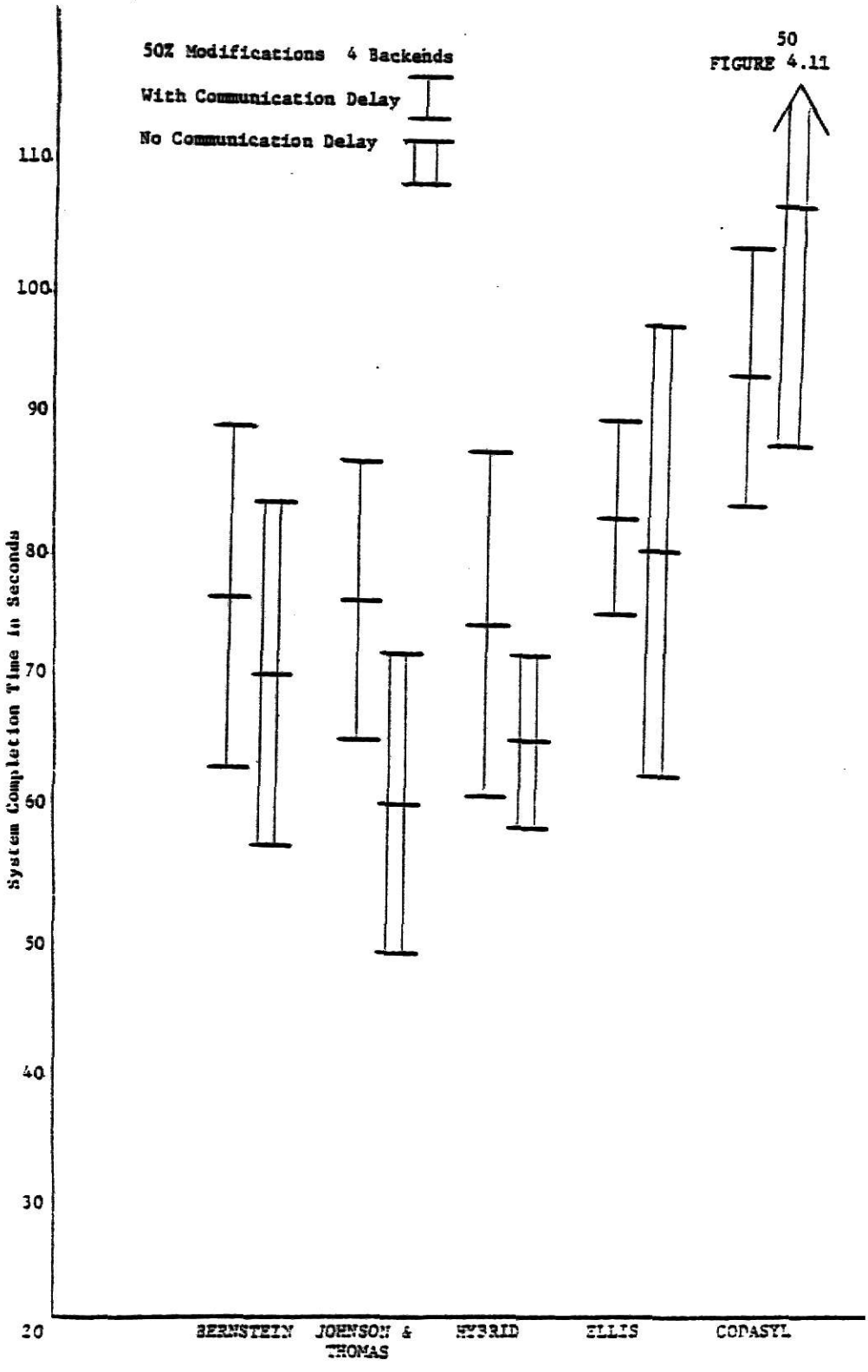
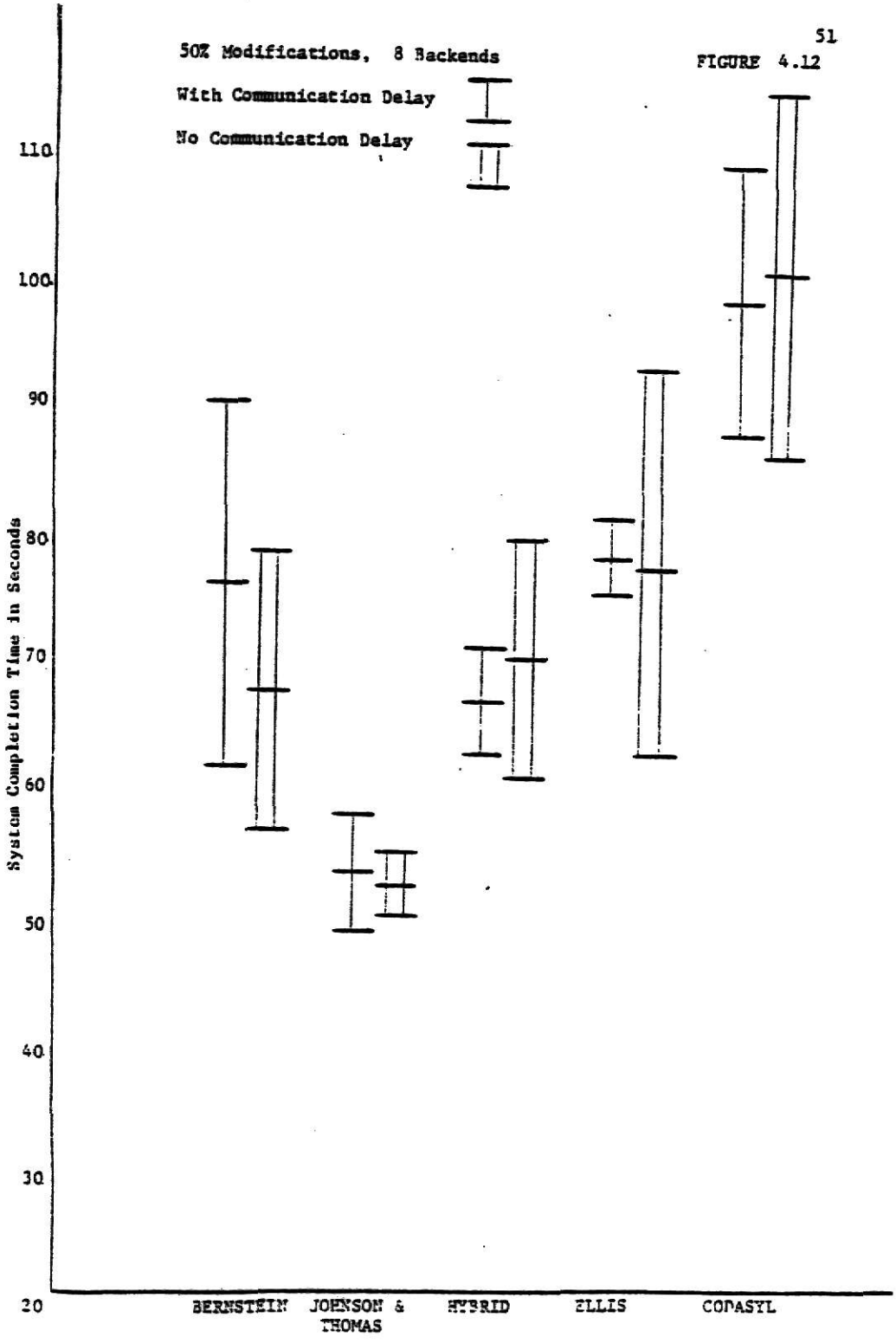

50% Modifications 4 Backends
With Communication Delay
No Communication Delay
System Completion Time in Seconds
110
100
90
80
70
60
50
40
30
20
BERNSTEIN
JOHNSON & THOMAS
HYBRID
ELLIS
CODASYL

FIGURE 4.12


50% Modifications, 8 Backends
With Communication Delay
No Communication Delay
System Completion Time in Seconds
110
100
90
80
70
60
50
40
30
20
BERNSTEIN
JOHNSON & THOMAS
HYBRID
ELLIS
CODASYL

4) Increasing the channel speed has only a minimal affect on increasing job throughput.

4.3 ANALYSIS OF CHANGE WITHIN A MODEL

Examining the changes in performance times of a model as the environment is changed can afford some interesting insights. For example, Bernstein's model shows very little change except for the runs with 50% modification requests which needed more time. The Johnson and Thomas model is stable for 20% and 35% modification requests, but performs oddly on 50% modification requests with eight backends when it requires significantly less time (53809). The Hybrid model is quite stable except when 50% of the commands are modifications. In that environment it performs very slow (82643) with two backends, somewhat slow (74143) with four backends, and at its stable rate (66494) with eight backends.

The Ellis model shows little change except in the 50% modification requests job mix where it takes longer to process the requests. Finally, the Codasyl model shows dramatic increases in processing time directly related to the percent of modification requests. For 20% and 35% modification requests job mix the Codasyl model processed the job stream quickest in the four backend configuration.

4.3 ANALYSIS OF BETWEEN MODEL PERFORMANCE

The original intention of this study was to find the one model that out performed the other models in keeping the database consistent using the least amount of system time. Because the three best models performed so closely, choosing one solely on performance times would not be prudent. In choosing among the three candidate models one should eliminate the Johnson and Thomas model because it does not guarantee consistency. An example will best explain it's consistency anomoly. Figure 4.13 pictures the following actions. Modification request A, which happens to be the deletion of record 1, enters the Host processor at timestamp 1000 and is assigned to the queue at Backend 1. Modification request B, an update of record 1, enters the Host processor at timestamp 1100 and is assigned to the queue at Backend 2. If modification B finishes first, Backend 2 sends its update to Backend 1 to make the databases consistent. Meanwhile, Backend 1 completes its deletion and sends it on to Backend 2. Backend 1 will fail in attempting to execute the update from Backend 2 because record 1 has been deleted. Backend 2 will ignore the deletion from Backend 1 because it has an older timestamp. The end result will be inconsistency, Backend 1 will have deleted record 1 and Backend 2 will have retained a copy of record 1.

Eliminating the Johnson and Thomas model for reasons of possible inconsistency leaves only the Hybrid model and Bernstein's model. The Hybrid model has the disadvantage of

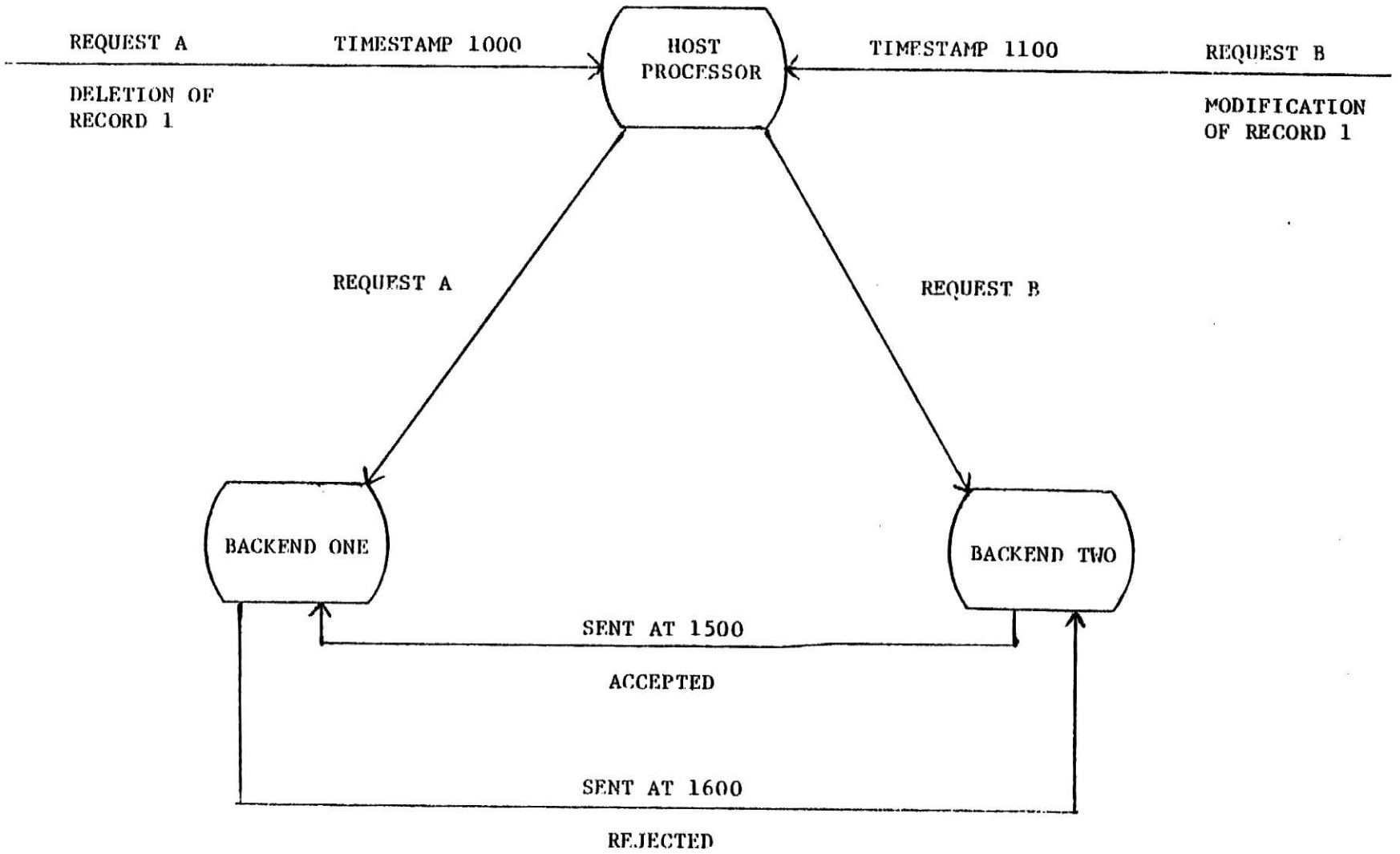

REQUEST A
TIMESTAMP 1000
HOST PROCESSOR
TIMESTAMP 1100
REQUEST B
DELETION OF RECORD 1
MODIFICATION OF RECORD 1
REQUEST A
REQUEST B
BACKEND ONE
BACKEND TWO
SENT AT 1500
ACCEPTED
SENT AT 1600
REJECTED

FIGURE 4.13

having never been formally proven. It also showed slower mean performance times although this figure was not significant. To test if the latter disadvantage could be corrected the Hybrid II model was tried. As explained in Chapter 3, the only difference between the Hybrid and the Hybrid II model was in the assignment of modification requests. The Hybrid II model assigned modification requests to the primary backend responsible for a granule of the database. This modification for the configurations where the jcb mix was 50% modification requests produced the results presented in Figure 4.14. Although the comparison shows the Hybrid II model as executing the requests faster than the Hybrid model, the difference was not significant. Since the differences in performance between the Hybrid models and the Berstein model are not significant, this will not be considered in the decision choosing the best overall model.

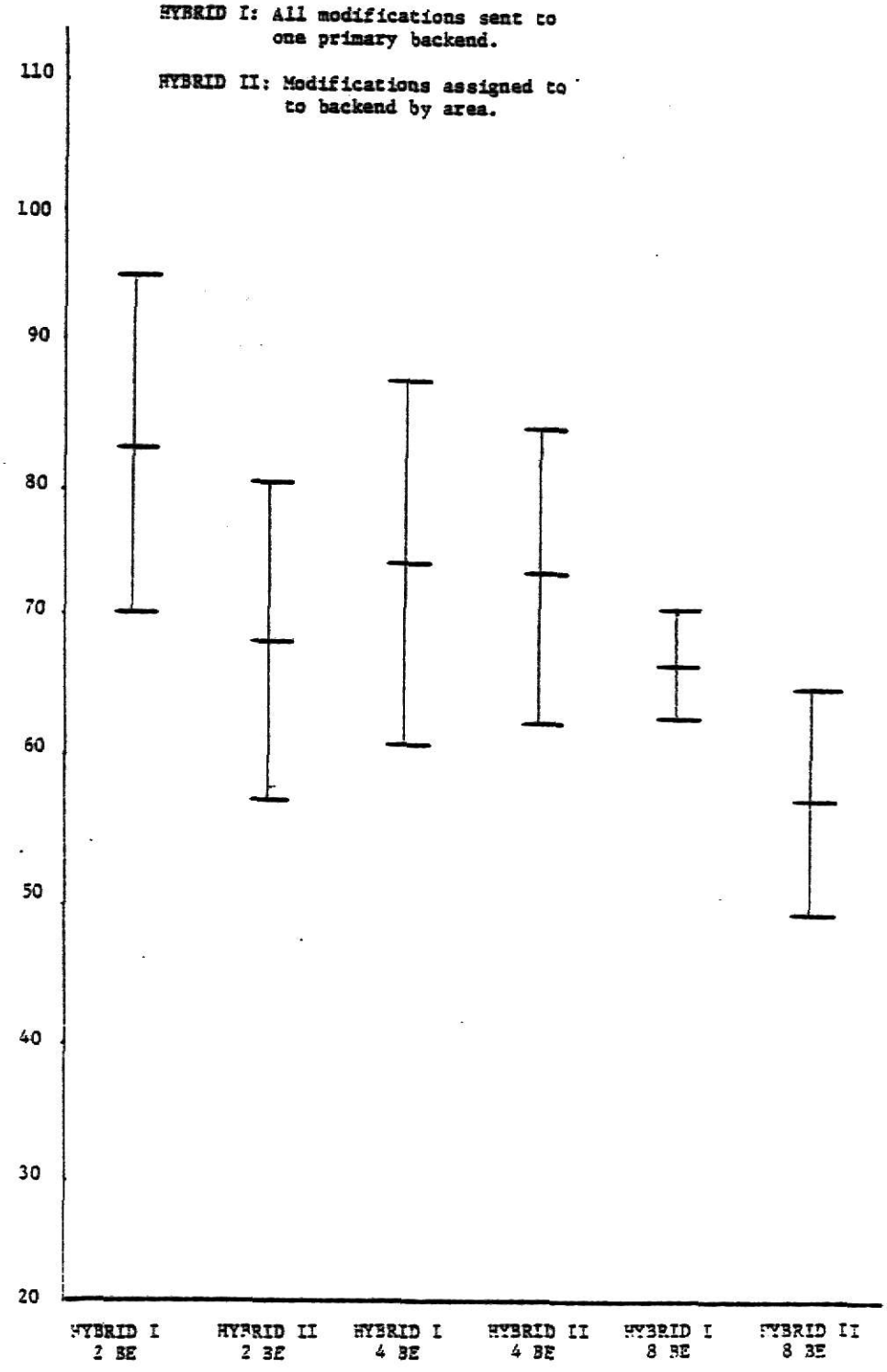
The Hybrid model's disadvantage in not having been proven is offset by the advantage that it is easy to understand and simple to implement. Bernstein's model, on the other hand is marred by its complexity. Its major advantage is that it has been proven. Therefore, in choosing between the two models, the designer is given the option of choosing a simple unproven model or a complex proven model. The author's personal choice would be to choose the simpler model to reduce the risk of software failure.

50% Modifications, Both Hybrid Models

FIGURE 4.14

HYBRID I: All modifications sent to one primary backend.

HYBRID II: Modifications assigned to to backend by area.



CHAPTER 5

CONCLUSION

5.1 SUMMARY

The simulation results support previous findings and break new ground in comparing the relative performance of five consistency algorithms for redundant databases. The results support a finding by Maryanski and Kreimer(8) that the addition of backend processors will increase system throughput so long as the system is heavily utilized. The results also indicate that the percent of modification requests present in the job stream will affect performance presumably because processing of a modification command blocks out processing of any other commands by that processor. It was also found that decreasing the channel speed had only a minimal affect on increasing job throughput.

The study drew conclusions by comparison of the performance of the five models in the different processing environments. The results found three of the models (Bernstein's model, Johnson and Thomas's model and the Hybrid model) performed significantly better than both the Ellis model and the Codasyl model. The difference between the three best models was not significant.

In choosing the ideal algorithm from the three best models, the Johnson and Thomas model was eliminated due to its proven possibility for inconsistency. The Bernstein model was critized for its complexity and the Hybrid model was critized for having never been formallly proven. The choice for the designer is between a simple but unproven model (the Hybrid model) and a complex but proven model (the Bernstein model). The author expressed a preference for the Hybrid model.

5.2 FUTURE CONSIDERATIONS

In this study a large and complex system was simulated. Due to time and efficiency restraints, some assumptions and simplifications in the models were made. Although most of the results seem reasonable, there remains a question as to their applicability in the real world. How might the accuracy of the results be evaluated and improved? There are two areas where more study could measure the accuracy of the results.

1) An actual multiprocessor backend system should be studied in detail. Measurements should be made in both hardware and software to determine utilization, queue lengths, job mixes, and other parameters which could be used in the models.

2) One or more of the models should be implemented and additional measurements taken. The measured results should

be compared with the simulation results to test the accuracy of the models.

REFERENCES

1. Bernstein, P. A., Rothnie, J. B., Goodman, N., Papadimitrou, C. A., 'The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redudant Case)', IEEE Transactions of Software Engineering, Vol. SE-4, NO.3, May 1978, pp158-167

2. Canady, R. H., Harrison, R. D., Ivie, E. L., Ryder, J. L., Wehr, L. A.,'A Backend Computer for Data Base Management', Communications of the ACM, Vol. 17, 10, 1974, pp575-582

3. Ellis, C. A.,'A Robust Algorithm for Updating Duplicate Databases', Proceedings of the Second Annual Berkeley Workshop on Distributed Data Management and Computer Networks, 1977, pp146-158

4. Everest, G., Personal Communication, Fall, 1977

5. General Purpose Simulation System V User's Manual (SH20-0866) IBM Corporation Third Edition September, 1977

6. Johnson, P. R., and Thomas, R. H., 'The Maintenance of Duplicate Databases', Network Working GRoup RFC 677, 1977

7. Maryanski, F. J., Wallentine, V. E., Fisher, P. S., Calhoun, M. A.,'Distributed Data Base Management System Using Minicomputers', in Infotech State-of-the-Art Report, Minis, Midis, and Mainframes, April 1978

8. Maryanski, F. J.,and Kreimer, D. E., 'Effects of Distributed Processing in a Data Processing Environment', Proceedings of the 11th Annual Simulation Symposium, March 1978, pp183-197

9. Macyanski, F. J., 'A Survey of Developments in Distributed Data Base Management Systems', IEEE Computer (11,2) February 1978, pp28-38

```
BLOCK
NUMBER  *LOC    OPERATION  A,B,C,D,E,F,G,H,I          COMMENTS
                SIMULATE
                RMULT      92576
                RMULT      14523
          *
          *       THE BERNSTEIN MODEL
          *
          * **    ENTITY DEFINITIONS
          *
          *   PARAMETERS
          *
          *       PH1: TERMINAL ASSIGNMENT NUMBER
          *       PF1: HALFWORD SAVEVALUE WHOSE NUMBER IS 10+PH5
          *       PH2: NUMBER OF I/O REQUESTS NECESSARY FOR EACH USER RWQUEST
          *       PF2: THR SPECIFIC RECORD NUMBER
          *       PH3: SIZE OF MESSAGE BUFFER
          *       PF3: THE ABSOLUTE CLOCK TIME
          *       PH4: MARKED 0 IF READ REQUEST
          *                   1 IF PRIMARY UPDATE REQUEST
          *                   2 IF SECONDARY UPDATE REQUEST
          *       PF4: MARKED 0 IF UPDATE REQUEST HAS NOT BEEN SPOOLED
          *                   1 IF UPDATE REQUEST HAS BEEN SPOOLED.
          *       PH5: ID NUMBER OF THE BACKEND MACHINE BEING USED
          *       PH6: NUMBER OF BACKEND MACHINES USED IN THE SIMULATION
          *       PH7: A COUNTER
          *       PH8: NUMBER OF REQUESTS MADE BY THE USER
          *       PH9: CHANNEL ID USED BETWEEN TWO BACKEND MACHINES
          *       PF11: TRANSMISSION SPEED TO CHANNEL IN BITS/SEC
          *       PF12: CHANNEL TRANSMISSION SPEED IN BITS/SEC
          *       PH13: TIME USED BY HINT OR BINT
          *       PH14: TIME USED BY THE MESSAGE SYSTEM
          *       PH15: ID NUMBER OF THE BE THAT THE SECONDARY UPDATE IS SENT TO
          *
          *   STORAGES REPRESENT PARTITIONS IN EACH BACKEND
          *
          *   FACILITIES
          *
          *       1 - 4:  CPU'S OF THE FOUR BACKEND MACHINES
          *       11,22,33,44:  CHANNELS FROM BACKEND TO DEVICE
          *       12->14: CHANNELS FROM BE1 TO BE2,BE3,AND BE4
          *       21,23,24: CHANNELS FROM BE2 TO BE1,BE3,AND BE4
          *       31,32,34: CHANNELS FROM BE3 TO BE1,BE2,AND BE4
          *       41->43: CHANNELS FROM BE4 TO BE1,BE2,AND BE3
          *       91->94:  CHANNELS BETWEEN HOST AND BE
          *       99: HOST CPU
          *       100: TERMINAL TO HOST CHANNEL
          *       101->109: TERMINALS CONNECTED TO HOST
          *
          *   HALFWORD SAVEVALUES
          *       XH1->XH8: CURRENT TABLE POINTERS FOR BE1->BE8
          *       XH11->XH18: LAST TABLE POINTERS FOR BE1->BE8
          *
          *   DEFINITIONS
          *       UPCAT1: PRIMARY UPDATE
          *       UPCAT2: SECONDARY UPDATE
          *       LAST: ROW OF MATRIX LAST USED IN UPDATING
```

```
*         CURRENT: NUMBER OF UPDATES SPOOLED TO THAT BE
*
*    FULLWORD MATRIX
*         1: UPDATE TABLE FOR BE1
*         2: UPDATE TABLE FOR BE2
*         3: UPDATE TABLE FOR BE3
*         4: UPDATE TABLE FOR BE4
*         5: UPDATE TABLE FOR BE5
*         6: UPDATE TABLE FOR BE6
*         7: UPDATE TABLE FOR BE7
*         8: UPDATE TABLE FOR BE8
*
*
 1      MATRIX      MX,150,4
 2      MATRIX      MX,150,4
 3      MATRIX      MX,150,4
 4      MATRIX      MX,150,4
 5      MATRIX      MX,150,4
 6      MATRIX      MX,150,4
 7      MATRIX      MX,150,4
 8      MATRIX      MX,150,4
        STORAGE     S1-S8,2
 MINUS  VARIABLE    PH6-1
 TRMNL  FVARIABLE   PH3*8/PF11*1000
 CHANL  FVARIABLE   PH3*8/PF12*1000
 LAST   VARIABLE    10+PH5
 INCRE  VARIABLE    PH7+1
 BECNL  VARIABLE    10*PH5+PH15
 TYPE   FUNCTION    RN2,D2
.80,0/1.00,1
*   20% UPDATE
 TERM   FUNCTION    RN2,D8
.125,101/.25,102/.375,103/.5,104/.625,105/.75,106/.875,107/1.0,108
 RECRD  FUNCTION    RN2,D10
.10,1/.20,2/.30,3/.40,4/.50,5/.60,6/.70,7/.80,8/.90,9/1.0,10
 REQNO  FUNCTION    RN2,D10
.10,1/.20,2/.30,4/.40,5/.50,7/.60,9/.70,11/.80,12/.90,13/1.00,14
 LNGTH  FUNCTION    RN2,D2
.90,128/1.00,256
*   90% FIT IN ONE BUFFER
 HOBE   FUNCTION    PH5,D8            HOST TO BE CHANNELS
1,91/2,92/3,93/4,94/5,95/6,96/7,97/8,98
 BEDEV  FUNCTION    PH5,D8            BE TO DEVICE CHANNELS
1,11/2,22/3,33/4,44/5,55/6,66/7,77/8,88
 USREQ  FUNCTION    RN2,D9            # OF REQUESTS BY A USER
.10,1/.20,3/.40,5/.50,7/.60,9/.70,11/.80,13/.90,15/1.00,17
*
 RNDM   FUNCTION    RN2,D4
.25,1/.50,2/.75,3/1.00,4
*
        INITIAL     XH1-XH8,0        INITIALIZE SAVEVALUES 1-10 TO ZERO
        INITIAL     XF1-XF8,0         INITIALIZE XF1-8 TO ZERO
        INITIAL     XH11-XH14,0      INITIALIZE SAVEVALUES 11-14 TO ZERO
*
1       GENERATE    1500,50,,15,,15PF,15PH  GENERATE 15 TRANSX, AVE 1/1.5 SEC
2       PRIORITY    1
3       ASSIGN      1,FNSTERM,PH     ASSIGN TERMINAL # IN PH1
```

```
4               ASSIGN      6,4,PH              NUMBER OF BACKENDS IN SIMULATION
5               ASSIGN      6,2,PF              PRIMARY/SECONDARY UPDATE CODE.INIT 2 OR R
6               ASSIGN      8,FN$USREQ,PH       STORE # OF USER'S COMMANDS IN PH8
7               ASSIGN      11,50000,PF            TRANSMISSION SPEED TO CHANNEL
8               ASSIGN      12,50000,PF            TRANSMISSION SPEED OF CHANNEL
9               ASSIGN      13,3,PH             ADVANCE TIME FOR HINT OR BINT
10              ASSIGN      14,5,PH             ADVANCE TIME FOR MESSAGE SYSTEM
11              QUEUE       PH1                 QUEUE FOR TERMINAL
12              SEIZE       PH1                 SEIZE THE TERMINAL
13              DEPART      PH1                 DEPART QUEUE FOR TERMINAL
                *
                *               MAIN LOOP FOR USER REQUESTS
                *
14              MORE  ADVANCE     1000,500          TIME TAKEN TO TYPE REQUEST
15                    ASSIGN      8-,1,PH           DECREMENT # OF REQUESTS MADE BY USER
16                    QUEUE       100               QUEUE HOST/TERMINAL CHANNEL
17                    SEIZE       100               TERMINAL-HOST CHANNEL
18                    DEPART      100               LEAVE QUEUE HOST-TERMINAL CHANNEL
19                    ADVANCE     V$TRMNL           LINE TRANSMISSION
20                    RELEASE     100               RELEASE CHANNEL
21                    QUEUE       99                QUEUE FOR CPU
22                    SEIZE       99                SEIZE HOST CPU
23                    DEPART      99
24                    ASSIGN      2,FN$REQNO,PH     ASSIGN # OF IO REQUESTS IN PH 2
25                    ASSIGN      3,FN$LNGTH,PH     LENGTH OF BUFFER TRANSMISSION
26                    ASSIGN      4,FN$TYPE,PH      ASSIGN UPDATE OR READ
27                    ADVANCE     PH13              HINT
28                    ADVANCE     PH14              MESSAGE SYSTEM
29                    TEST E      PH4,1,*+2         IF UPDATE MARK PF6 A 1
30                    ASSIGN      6,1,PF
31                    ASSIGN      2,FN$RECRD,PF     MAKE RECORD ASSIGNMENT
32                    ASSIGN      3,C1,PF           MAKE CLOCK TIME ASSIGNMENT
33                    ASSIGN      5,1,PH
                *
                *             CHECK FOR A FREE MACHINE
                *
34              LOOPA RELEASE     99                FREE HOST CPU
35                    QUEUE       FN$HOBE           WAIT FOR HOST-BE CHANNEL
36                    SEIZE       FN$HOBE           SEIZE HOST-BE CHANNEL
37                    DEPART      FN$HOBE
38                    ADVANCE     V$CHANL
39                    ADVANCE     1                 TIME TAKEN TO CHECK IF FREE
40                    GATE SNE    PH5,BEDB          GO TO BEDB IF BE FREE
41                    RELEASE     FN$HOBE           RETURN CHANNEL
42                    QUEUE       99
43                    SEIZE       99                SEIZE HOST CPU
44                    DEPART      99
45                    ASSIGN      5+,1,PH
46                    TEST LE     PH5,PH6,FULL      SEND TO FULL IF ALL BE'S ARE BUSY
47                    TRANSFER    ,LOOPA
48              FULL  ASSIGN      5,FN$RNOM,PH      RANDOMLY ASSIGN BE MACHINE
49                    ADVANCE     1                 ASSIGNMENT TIME
50                    RELEASE     99                RELEASE THE HOST PROCESSOR
                *
                *             BACKEND HAS BEEN SELECTED
                *
51                    QUEUE       FN$HOBE
```

```
52              SEIZE       FN$HOBE
53              DEPART      FN$HOBE
54              ADVANCE     V$CHANL
55        BED8  RELEASE     FN$HOBE
56              ADVANCE     PH13                BINT
57              QUEUE       PH5                 QUEUE FOR BE PARTITION
58              ENTER       PH5                 ENTER BE PARTITION
59              DEPART      PH5
60        BEGIN TEST E      PH4,1,*+2           CHK IF UPDATE
61              PRIORITY    3                   INCREASE PROIRITY FOR UPDATES
62              SEIZE       PH5                 SEIZE BE CPU
63              TRANSFER    ,REQ1               ENTER REQUEST IN REQUEST TABLE
          *
64         VCTE TEST E      PH4,1,*+2           IF NOT AN UPDATE, SKIP 2 LINES
65              TEST E      PF6,0,UPDT1         IF PRIMARY UPDATE GO TO UPDT1
66         LABL ASSIGN      7,0,PH              PH7 IS A COUNTER
          *
          *              PROCESSING OF IO REQUESTS
          *
67         LOOP8 TEST NE    PH2,PH7,FINIO       IF IO DONE GO TO FINIO
68              TEST NE     PH4,1,*+2           IF UPDATE DON'T RELEASE DATABASE
69              RELEASE     PH5                 RETURN BE CPU
70              QUEUE       FN$BEDEV
71              SEIZE       FN$BEDEV            SEIZE BE DEVICE FOR IO
72              DEPART      FN$BEDEV
73              ADVANCE     52                  TIME FOR EACH IO
74              RELEASE     FN$BEDEV            RELEASE DEVICE
75              TEST NE     PH4,1,*+2           IF UPDATE, BE IS ALREADY SEIZED
76              SEIZE       PH5                 SEIZE BE CPU
77              ADVANCE     1                   PROCESSING TIME
78              ASSIGN      7+,1,PH             INCREMENT COUNTER
79              TRANSFER    ,LOOP8
          *
          *              IO COMPLETED
          *
80         FINIO TEST NE    PF6,0,UPDT2         IF SECONDARY UPDATE GO TO UPDT2
81              TEST E      PF6,1,*+2           IF NOT A PRIMARY UPDATE, SKIP AROUND
82              SAVEVALUE   PH5+,1,XF           RELEASE THE SECONDARY MODIFICATIONS
          *
          *              RETURN INFORMATION TO THE HOST AND TERMINALS
          *
83         READ1 MSAVEVALUE PH5,PF9,4,0,MX      MARK REQUEST AS BEING COMPLETE
84              ADVANCE     PH13                BINT
85              RELEASE     PH5
86              LEAVE       PH5                 LEAVE PARTITION IN BE
87              QUEUE       FN$HOBE             WAIT FOR HOST-BE CHANNEL
88              SEIZE       FN$HOBE
89              DEPART      FN$HOBE
90              ADVANCE     V$CHANL
91              ADVANCE     PH14                MESSAGE SYSTEM
92              RELEASE     FN$HOBE
93              QUEUE       99                  QUEUE FOR HOST
94              PREEMPT     99,PR               PREEMPT HOST CPU
95              DEPART      99
96              ADVANCE     PH13                HINT
97              RETURN      99
98              QUEUE       100                 WAIT FOT CPU-TERMINAL CHANNEL
```

```
99              PREEMPT     100,PR
100             DEPART      100
101             ADVANCE     VSTRMNL
102             RETURN      100
103             TEST E      PH8,0,MORE          MORE REQUESTS BY THIS USER?
104             RELEASE     PH1                 RELEASE OCCUPIED TERMINAL
105       ENDIT TERMINATE   1
          *
          *               ENTER REQUESTS IN MODIFICATION TABLE
          *
106       REQ1  SAVEVALUE   PH5+,1,XH           INCREMENT # OF REQUESTS MADE
107             ASSIGN      9,XH*PH5,PF         STORE LAST POINTER NUMBER IN PF9
108             MSAVEVALUE  PH5,PF9,4,1,MX      MARK REQUEST AS PENDING
109             TEST NE     PH5,1,ONE           DETERMINE BE YOU ARE REQUESTING ON
110             TEST NE     PH5,2,TWO           SO YOU CAN UPDATE APPROPRIATE TABLE.
111             TEST NE     PH5,3,THREE
112             TEST NE     PH5,4,FOUR
113             TEST NE     PH5,5,FIVE
114             TEST NE     PH5,6,SIX
115             TEST NE     PH5,7,SEVEN
116             TEST NE     PH5,8,EIGHT
117       EIGHT MSAVEVALUE  8,XH8,1,PH2,MX  STORE THE NUMBER OF IO REQUESTS
118             MSAVEVALUE  8,XH8,2,PF2,MX  STORE THE RECORD NUMBER
119             MSAVEVALUE  8,XH8,3,PF3,MX  STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
120             TRANSFER    ,SKIP1
121       SEVEN MSAVEVALUE  7,XH7,1,PH2,MX  STORE THE NUMBER OF IO REQUESTS
122             MSAVEVALUE  7,XH7,2,PF2,MX  STORE THE RECORD NUMBER
123             MSAVEVALUE  7,XH7,3,PF3,MX  STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
124             TRANSFER    ,SKIP1
125       SIX   MSAVEVALUE  6,XH6,1,PH2,MX  STORE THE NUMBER OF IO REQUESTS
126             MSAVEVALUE  6,XH6,2,PF2,MX  STORE THE RECORD NUMBER
127             MSAVEVALUE  6,XH6,3,PF3,MX  STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
128             TRANSFER    ,SKIP1
129       FIVE  MSAVEVALUE  5,XH5,1,PH2,MX  STORE THE NUMBER OF IO REQUEXTS
130             MSAVEVALUE  5,XH5,2,PF2,MX  STORE THE RECORD NUMBER
131             MSAVEVALUE  5,XH5,3,PF3,MX  STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
132             TRANSFER    ,SKIP1
133       FOUR  MSAVEVALUE  4,XH4,1,PH2,MX  STORE THE NUMBER OF IO REQUEXTS
134             MSAVEVALUE  4,XH4,2,PF2,MX  STORE THE RECORD NUMBER
135             MSAVEVALUE  4,XH4,3,PF3,MX  STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
136             TRANSFER    ,SKIP1
137       THREE MSAVEVALUE  3,XH3,1,PH2,MX  STORE THE NUMBER OF IO REQUESTS
138             MSAVEVALUE  3,XH3,2,PF2,MX  STORE THE RECORD NUMBER
139             MSAVEVALUE  3,XH3,3,PF3,MX  STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
140             TRANSFER    ,SKIP1
141       TWO   MSAVEVALUE  2,XH2,1,PH2,MX  STORE THE NUMBER OF IO REQUESTS
142             MSAVEVALUE  2,XH2,2,PF2,MX  STORE THE RECORD NUMBER
143             MSAVEVALUE  2,XH2,3,PF3,MX  STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
144             TRANSFER    ,SKIP1
145       ONE   MSAVEVALUE  1,XH1,1,PH2,MX  STORE THE NUMBER OF IO REQUESTS
146             MSAVEVALUE  1,XH1,2,PF2,MX  STORE THE RECORD NUMBER
147             MSAVEVALUE  1,XH1,3,PF3,MX  STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
148             TRANSFER    ,SKIP1
149       SKIP1 ADVANCE     1                 TIME TAKEN TO WRITE TO TABLE
150             TRANSFER    ,VOTE
          *
          *               ROUTINE CALLED FOR MODIFICATION REQUESTS
```

```
          *
151       UPDT1 TEST E      PF6,1,UPDT2       GO TO UPDT2 IF THIS IS NOT A PRIMARY UPDT
          *
152       SKIP5 ASSIGN      7,1,PH            A COUNTER
          *
153       SKIP6 TEST LE     PH7,PH6,VOTIN     IF COUNTR GT # OF BES, GO TO VOTEIN
154             TEST NE     PH7,PH5,ELSE
155             ASSIGN      15,PH7,PH         DATERMINE # OF BE TO BE UPDTED
156             TRANSFER    ,*+3
157       ELSE  ASSIGN      7+,1,PH
158             TRANSFER    ,SKIP6
159             ASSIGN      7+,1,PH           INCREMENT COUNTER
160             SPLIT       1,*+2
161             TRANSFER    ,SKIP6
          *
162             QUEUE       V$BECNL           QUEUE FOR BE TO BE CHANNEL
163             SEIZE       V$BECNL           SEIZE THE BE TO BE CHANNEL
164             DEPART      V$BECNL           LEAVE THE BE TO BE CHANNEL QUEUE
165             ADVANCE     V$CHANL           TIME TAKEN TO SEND MESSAGE ACROSS THE CHA
166             RELEASE     V$BECNL           FREE THE BE TO BE CHANNEL
167             ASSIGN      5,XH*PH15,PF      ASSIGN INTO PF5 # OF LAST ENTRY IN TABLE
          *
          *               SEARCH MODIFICATION TABLE
          *
168       LOOPM TEST NE     PF5,0,VOTEA       IF THERE ARE NO ENTRIES  VOTE ACCEPT
169             TEST L      MX*PH5(PF9,3),MX*PH15(PF5,3),VOTEA    TIMESTAMP LESS-VOTE
170             TEST E      MX*PH5(PF9,2),MX*PH15(PF5,2),DECRE    IF RECRD # DIFFRENT-
171       THERE TEST E      MX*PH15(PF5,4),0,WAIT    IF REQUEST PENDING - WAIT
          *
172       VOTEA ADVANCE     10                VOTING TIME
173             QUEUE       V$BECNL           QUEUE FOR BE TO BE CHANNEL
174             SEIZE       V$BECNL           SEIZE THE BE TO BE CHANNEL
175             DEPART      V$BECNL           LEAVE THE BE TO BE CHANNEL QUEUE
176             ADVANCE     V$CHANL           TIME TAKEN TO SEND MESSAGE ACROSS THE CHA
177             RELEASE     V$BECNL           FREE THE BE TO BE CHANNEL
178             SAVEVALUE   PH5+,1,XF         INCREMENT VOTE TALLYIER
179             TEST E      XF*PH5,PH6        HOLD TILL ALL VOTES IN
180             ADVANCE     10                TIME TAKEN TO 'WRITE' OK MESSAGE
181             SAVEVALUE   PH5,0,XF          SET VOTE TALLYIER BACK TO 0
182             ADVANCE     1                 DELAY NEEDED TO ASSEMBLE THE REQUESTS
183             SAVEVALUE   PH5+,1,XF         INCREMENT WAIT COUNTER
184             TEST E      XF*PH5,PH6        WAIT UNTIL PRIMARY UPDATE IS COMPLETE
185             QUEUE       V$BECNL           QUEUE FOR BE TO BE CHANNEL
186             SEIZE       V$BECNL           SEIZE THE BE TO BE CHANNEL
187             DEPART      V$BECNL           LEAVE THE BE TO BE CHANNEL QUEUE
188             ADVANCE     V$CHANL           TIME TAKEN TO SEND MESSAGE ACROSS THE CHA
189             RELEASE     V$BECNL           FREE THE BE TO BE CHANNEL
190             ASSIGN      5,PH15,PH         ASSIGN BE # INTO PH5
191             ASSIGN      6,0,PF            MARK AS SECONDARY UPDATE
192             QUEUE       PH5               QUEUE FOR BE PARTITION
193             ENTER       PH5               ENTER BE PARTITION
194             DEPART      PH5
195             TRANSFER    ,BEGIN            GO TO BEGIN AND BEGIN SECONDARY UPDATING
196       WAIT  ADVANCE     5                 TIME OUT 5 MSEC
197             TRANSFER    ,THERE
198       DECRE ASSIGN      5-,1,PF           DECREMENT TABLE POINTER
199             TRANSFER    ,LOOPM
```

```
200      VCTIN SAVEVALUE  PH5+,1,XF          INCREMENT VOTE TALLYIER
201            TEST E     XF*PH5,PH6         HOLD TILL ALL VOTES COUNTED
202            TRANSFER   ,LABL
         *
203      UPOT2 MSAVEVALUE PH15,XH*PH15,4,0,MX      MARK REQUEST AS COMPLETED
204            LEAVE      PH5                LEAVE BE PARTITION
205            RELEASE    PH5
206            ASSEMBLE   VSMINUS            ASSEMBLE ALL SECONDARY UPDATES
207            TRANSFER   ,ENDIT
               START      800,,400
               NOXREF
               END
```

**** ASSEMBLY TIME = .04 MINUTES ****

```
BLOCK
NUMBER  #LOC    OPERATION   A,B,C,D,E,F,G,H,I          COMMENTS
                SIMULATE
                RMULT       14523
                RMULT       37
*
*         JOHNSON & THOMAS
*
* **    ENTITY DEFINITIONS
*
*    PARAMETERS
*
*         PH1: TERMINAL ASSIGNMENT NUMBER
*         PF1: HALFWORD SAVEVALUE WHOSE NUMBER IS 10+PH5
*         PH2: NUMBER OF I/O REQUESTS NECESSARY FOR EACH USER RWQUEST
*         PF2: THR SPECIFIC RECORD NUMBER
*         PH3: SIZE OF MESSAGE BUFFER
*         PF3: THE ABSOLUTE CLOCK TIME
*         PH4: MARKED 0 IF READ REQUEST
*                     1 IF PRIMARY UPDATE REQUEST
*                     2 IF SECONDARY UPDATE REQUEST
*         PF4: MARKED 0 IF UPDATE REQUEST HAS NOT BEEN SPOOLED
*                     1 IF UPDATE REQUEST HAS BEEN SPOOLED
*         PH5: ID NUMBER OF THE BACKEND MACHINE BEING USED
*         PH6: NUMBER OF BACKEND MACHINES USED IN THE SIMULATION
*         PH7: A COUNTER
*         PH8: NUMBER OF REQUESTS MADE BY THE USER
*         PH9: CHANNEL ID USED BETWEEN TWO BACKEND MACHINES
*         PF11: TRANSNISSION SPEED TO CHANNEL IN BITS/SEC
*         PF12: CHANNEL TRANSMISSION SPEED IN BITS/SEC
*         PH13: TIME USED BY HINT OR BINT
*         PH14: TIME USED BY THE MESSAGE SYSTEM
*         PH15: ID NUMBER OF THE BE THAT THE SECONDARY UPDATE IS SENT TO
*
*    FACILITIES
*
*         1 - 4:  CPU'S OF THE FOUR BACKEND MACHINES
*         11,22,33,44:  CHANNELS FROM BACKEND TO DEVICE
*         12->14: CHANNELS FROM BE1 TO BE2,BE3,AND BE4
*         21,23,24: CHANNELS FROM BE2 TO BE1,BE3,AND BE4
*         31,32,34: CHANNELS FROM BE3 TO BE1,BE2,AND BE4
*         41->43: CHANNELS FROM BE4 TO BE1,BE2,AND BE3
*         91->94:  CHANNELS BETWEEN HOST AND BE
*         99: HOST CPU
*         100: TERMINAL TO HOST CHANNEL
*         101->109: TERMINALS CONNECTED TO HOST
*
*    HALFWORD SAVEVALUES
*         XH1->XH4: CURRENT TABLE POINTERS FOR BE1->BE4
*         XH11->XH14: LAST TABLE POINTERS FOR BE1->BE4
*
*    DEFINITIONS
*         UPDAT1: PRIMARY UPDATE
*         UPDAT2: SECONDARY UPDATE
*         LAST: ROW OF MATRIX LAST USED IN UPDATING
*         CURRENT: NUMBER OF UPDATES SPOOLED TO THAT BE
*
```

```
*
*    HALFWORD MATRIX
*         1: UPDATE TABLE FOR BE1
*         2: UPDATE TABLE FOR BE2
*         3: UPDATE TABLE FOR BE3
*         4: UPDATE TABLE FOR BE4
*         5: UPDATE TABLE FOR BE5
*         6: UPDATE TABLE FOR BE6
*         7: UPDATE TABLE FOR BE7
*         8: UPDATE TABLE FOR BE8
*
 1      MATRIX      MX,50,3
 2      MATRIX      MX,50,3
 3      MATRIX      MX,50,3
 4      MATRIX      MX,50,3
 5      MATRIX      MX,50,3
 6      MATRIX      MX,50,3
 7      MATRIX      MX,50,3
 8      MATRIX      MX,50,3
*
*    STORAGES REPRESENT PARTITIONS IN EACH BACKEND
*
        STORAGE     S1-S8,2
 TRMNL FVARIABLE   PH3*8/PF11*1000
 CHANL FVARIABLE   PH3*8/PF12*1000
 LAST  VARIABLE    10+PH5
 INCRE VARIABLE    PH7+1
 BECNL VARIABLE    10*PH5+PH15
 TYPE  FUNCTION    RN2,D2
.80,0/1.00,1
*         20%  UPDATE
 TERM  FUNCTION    RN2,D8
.125,101/.25,102/.375,103/.5,104/.625,105/.75,106/.875,107/1.0,108
 RECRD FUNCTION    RN2,D10
.10,1/.20,2/.30,3/.40,4/.50,5/.60,6/.70,7/.80,8/.90,9/1.0,10
 REQNO FUNCTION    RN2,D10
.10,1/.20,2/.30,4/.40,5/.50,7/.60,9/.70,11/.80,12/.90,13/1.00,14
 LNGTH FUNCTION    RN2,D2
.90,128/1.00,256
*   90% FIT IN ONE BUFFER
 HOBE  FUNCTION    PH5,D8            HOST TO BE CHANNELS
1,91/2,92/3,93/4,94/5,95/6,96/7,97/8,98
 BEDEV FUNCTION    PH5,D8            BE TO DEVICE CHANNELS
1,11/2,22/3,33/4,44/5,55/6,66/7,77/8,88
 USREQ FUNCTION    RN2,D9            # OF REQUESTS BY A USER
.10,1/.20,3/.40,5/.50,7/.60,9/.70,11/.80,13/.90,15/1.00,17
*
 RNDM  FUNCTION    RN2,D4
.25,1/.50,2/.75,3/1.00,4
*
        INITIAL     XH1-XH9,0       INITIALIZE SAVEVALUES 1-10 TO ZERO
        INITIAL     XH11-XH18,0     INITIALIZE SAVEVALUES 11-18 TO ZERO
*
1       GENERATE    1500,50,,15,,15PF,15PH  GENERATE 15 TRANSX, AVE 1/1.5 SEC
2       PRIORITY    1
3       ASSIGN      1,FN$TERM,PH     ASSIGN TERMINAL # IN PH1
4       ASSIGN      6,4,PH           NUMBER OF BACKENDS IN SIMULATION
```

```
5               ASSIGN      8,FNSUSREQ,PH     STORE # OF USER'S COMMANDS IN PH8
6               ASSIGN      11,50000,PF       TRANSMISSION SPEED TO CHANNEL
7               ASSIGN      12,50000,PF       TRANSMISSION SPEED OF CHANNEL
8               ASSIGN      13,5,PH           ADVANCE TIME FOR HINT OR BINT
9               ASSIGN      14,5,PH           ADVANCE TIME FOR MESSAGE SYSTEM
10              SAVEVALUE   9+,1,XH           INCREMENT USER COUNTER
11              TEST E      XH9,15,*+2        IS THIS THE LAST USER?
12              ASSIGN      9,1,PF            MARK AS LAST USER
13              QUEUE       PH1               QUEUE FOR TERMINAL
14              SEIZE       PH1               SEIZE THE TERMINAL
15              DEPART      PH1               DEPART QUEUE FOR TERMINAL
                *
                *           MAIN LOOP FOR USER REQUESTS
                *
16          MORE ADVANCE    1000,500          TIME TAKEN TO TYPE REQUEST
17              ASSIGN      8-,1,PH           DECREMENT # OF REQUESTS MADE BY USER
18              QUEUE       100               QUEUE HOST/TERMINAL CHANNEL
19              SEIZE       100               TERMINAL-HOST CHANNEL
20              DEPART      100               LEAVE QUEUE HOST-TERMINAL CHANNEL
21              ADVANCE     VSTRMNL           LINE TRANSMISSION
22              ADVANCE     VSCHANL           CHANNEL TRANSMISSION
23              RELEASE     100               RELEASE CHANNEL
24              QUEUE       99                QUEUE FOR CPU
25              SEIZE       99                SEIZE HOST CPU
26              DEPART      99
27              ASSIGN      2,FNSREQNO,PH     ASSIGN # OF IO REQUESTS IN PH 2
28              ASSIGN      3,FNSLNGTH,PH     LENGTH OF BUFFER TRANSMISSION
29              ASSIGN      4,FNSTYPE,PH      ASSIGN UPDATE OR READ
30              ADVANCE     PH13              HINT
31              ADVANCE     PH14              MESSAGE SYSTEM
32              TEST E      PH4,1,*+3         IF UPDATE MAKE RECORD & CLOCK TIME ASSIGN
33              ASSIGN      2,FNSRECRD,PF     MAKE RECORD ASSIGNMENT
34              ASSIGN      3,C1,PF           MAKE CLOCK TIME ASSIGNMENT
35              ASSIGN      5,1,PH
                *
                *         CHECK FOR A FREE MACHINE
                *
36          LOOPA RELEASE   99                FREE HOST CPU
37              QUEUE       FNSHOBE           WAIT FOR HOST-BE CHANNEL
38              SEIZE       FNSHOBE           SEIZE HOST-BE CHANNEL
39              DEPART      FNSHOBE
40              ADVANCE     VSCHANL
41              ADVANCE     1                 TIME TAKEN TO CHECK IF FREE
42              GATE SNE    PH5,BEDB          GO TO BEDB IF BE FREE
43              RELEASE     FNSHOBE           RETURN CHANNEL
44              QUEUE       99
45              SEIZE       99                SEIZE HOST CPU
46              DEPART      99
47              ASSIGN      5+,1,PH
48              TEST LE     PH5,PH6,FULL      SEND TO FULL IF ALL BE'S ARE BUSY
49              TRANSFER    ,LOOPA
50          FULL ASSIGN     5,FNSRNDM,PH      RANDOMLY ASSIGN BE MACHINE
51              ADVANCE     1                 ASSIGNMENT TIME
52              RELEASE     99                RELEASE THE HOST PROCESSOR
                *
                *         BACKEND HAS BEEN SELECTED
                *
```

```
53              QUEUE       FNSHOBE
54              SEIZE       FNSHOBE
55              DEPART      FNSHOBE
56              ADVANCE     V$CHANL
57        BED8  RELEASE     FNSHOBE
58              ADVANCE     PH13                BINT
59              QUEUE       PH5                 QUEUE FOR BE PARTITION
60              ENTER       PH5                 ENTER BE PARTITION
61              DEPART      PH5
62              TEST E      PH4,1,*+2
63              PRIORITY    3
64              SEIZE       PH5                 SEIZE THE BE CPU
65              ASSIGN      7,0,PH              PH7 IS A COUNTER
          *
          *               PROCESSING OF IO REQUESTS
          *
66         LOOP8 TEST NE    PH2,PH7,FINIO       IF IO DONE GO TO FINIO
67              TEST NE     PH4,1,*+2
68              RELEASE     PH5
69              QUEUE       FNSBEDEV
70              SEIZE       FNSBEDEV            SEIZE BE DEVICE FOR IO
71              DEPART      FNSBEDEV
72              ADVANCE     52                  TIME FOR EACH IO
73              TEST E      PH4,1,*+4           IS REQUEST A PRIMARY UPDATE?
74              TEST NE     PF4,1,*+3           HAS THIS UPDATE ALREADY BEEN SPOOLED?
75              ADVANCE     30                  IF NOT, SPOOL THE UPDATE
76              ASSIGN      4,1,PF              MARK THE UPDATE AS SPOOLED
77              RELEASE     FNSBEDEV            RELEASE DEVICE
78              TEST NE     PH4,1,*+2
79              SEIZE       PH5                 SEIZE THE BE CPU
80              ADVANCE     1                   PROCESSING TIME
81              ASSIGN      7+,1,PH             INCREMENT COUNTER
82              TRANSFER    ,LOOP8
          *
          *
          *               IO COMPLETED
          *
83         FINIO TEST NE    PH4,1,UPDT1         IF UPDATE, GO TO UPDT1-CHANGE TABLE
84              TEST E      PH4,2,READ          IF QUERY ONLY COMMAND, GO TO READ
          *
          *         REQUEST IS A SECONDARY MODIFICATION
          *
85         RECPU RELEASE    PH5
86              LEAVE       PH5                 RELEASE THE BE PARTITION
87              LEAVE       9                   LEAVE SECONDARY UPDATE STORAGE
          *
88         FINUP TEST G     S*PH5,1,UPDT2       IF BE PARTITION EMPTY GO TO UPDATES
89              TRANSFER    ,ENDIT              ELSE END TRANSACTION
          *
          *               RETURN INFORMATION TO THE HOST AND TERMINALS
          *
90         READ  ADVANCE    PH13                BINT
91              RELEASE     PH5
92              LEAVE       PH5                 LEAVE PARTITION IN BE
93              TEST E      S*PH5,0,*+2         IF PARTITIONS EMPTY, SPLIT
94              SPLIT       1,UPDT2             SPLIT AND SEND 1 TO UPDT2
95              QUEUE       FNSHOBE             WAIT FOR HOST-BE CHANNEL
```

```
96              SEIZE       FNSHOBE
97              DEPART      FNSHOBE
98              ADVANCE     VSCHANL
99              ADVANCE     PH14                MESSAGE SYSTEM
100             RELEASE     FNSHOBE
101             QUEUE       99                  QUEUE FOR HOST
102             PREEMPT     99,PR               PREEMPT HOST CPU
103             DEPART      99
104             ADVANCE     PH13                HINT
105             RETURN      99
106             QUEUE       100                 WAIT FOT CPU-TERMINAL CHANNEL
107             PREEMPT     100,PR
108             DEPART      100
109             ADVANCE     VSTRMNL
110             RETURN      100
111             TEST E      PH8,0,MORE          MORE REQUESTS BY THIS USER?
112             RELEASE     PH1                 RELEASE OCCUPIED TERMINAL
113             TEST E      PF9,1,*+2           SKIP AROUND IF NOT LAST USER
114             GATE SE     9                   NO ENTRY TILL ALL UPDATES HAVE BEEN COMPL
115     ENDIT   TERMINATE   1
                *
                *       ENTER MODIFICATION REQUESTS IN BE MODIFICATION TABLE
                *
116     UPDT1   SAVEVALUE   PH5+,1,XH           INCREMENT # OF UPDATES SPOOLED
117             TEST NE     PH5,1,ONE           DETERMINE BE YOU ARE UPDATING ON
118             TEST NE     PH5,2,TWO           SO YOU CAN UPDATE APPROPRIATE TABLE.
119             TEST NE     PH5,3,THREE
120             TEST NE     PH5,4,FOUR
121             TEST NE     PH5,5,FIVE
122             TEST NE     PH5,6,SIX
123             TEST NE     PH5,7,SEVEN
124             TEST NE     PH5,8,EIGHT
125     EIGHT   MSAVEVALUE  8,XH8,1,PH2,MX      STORE THE NUMBER OF IO REQUESTS
126             MSAVEVALUE  8,XH8,2,PF2,MX      STORE THE RECORD NUMBER
127             MSAVEVALUE  8,XH8,3,PF3,MX      STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
128             TRANSFER    ,SKIP1
129     SEVEN   MSAVEVALUE  7,XH7,1,PH2,MX      STORE THE NUMBER OF IO REQUESTS
130             MSAVEVALUE  7,XH7,2,PF2,MX      STORE THE RECORD NUMBER
131             MSAVEVALUE  7,XH7,3,PF3,MX      STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
132             TRANSFER    ,SKIP1
133     SIX     MSAVEVALUE  6,XH6,1,PH2,MX      STORE THE NUMBER OF IO REQUESTS
134             MSAVEVALUE  6,XH6,2,PF2,MX      STORE THE RECORD NUMBER
135             MSAVEVALUE  6,XH6,3,PF3,MX      STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
136             TRANSFER    ,SKIP1
137     FIVE    MSAVEVALUE  5,XH5,1,PH2,MX      STORE THE NUMBER OF IO REQUEXTS
138             MSAVEVALUE  5,XH5,2,PF2,MX      STORE THE RECORD NUMBER
139             MSAVEVALUE  5,XH5,3,PF3,MX      STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
140             TRANSFER    ,SKIP1
141     FOUR    MSAVEVALUE  4,XH4,1,PH2,MX      STORE THE NUMBER OF IO REQUEXTS
142             MSAVEVALUE  4,XH4,2,PF2,MX      STORE THE RECORD NUMBER
143             MSAVEVALUE  4,XH4,3,PF3,MX      STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
144             TRANSFER    ,SKIP1
145     THREE   MSAVEVALUE  3,XH3,1,PH2,MX      STORE THE NUMBER OF IO REQUESTS
146             MSAVEVALUE  3,XH3,2,PF2,MX      STORE THE RECORD NUMBER
147             MSAVEVALUE  3,XH3,3,PF3,MX      STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
148             TRANSFER    ,SKIP1
149     TWO     MSAVEVALUE  2,XH2,1,PH2,MX      STORE THE NUMBER OF IO REQUESTS
```

```
150             MSAVEVALUE  2,XH2,2,PF2,MX   STORE THE RECORD NUMBER
151             MSAVEVALUE  2,XH2,3,PF3,MX   STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
152             TRANSFER    ,SKIP1
153      ONE    MSAVEVALUE  1,XH1,1,PH2,MX   STORE THE NUMBER OF IO REQUESTS
154             MSAVEVALUE  1,XH1,2,PF2,MX   STORE THE RECORD NUMBER
155             MSAVEVALUE  1,XH1,3,PF3,MX   STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
156             TRANSFER    ,SKIP1
157      SKIP1  ADVANCE     1                TIME TAKEN TO WRITE TO TABLE
158             TRANSFER    ,READ
                *
                *       ROUTINE CALLED TO PERFORM SECONDARY MODIFICATION REQUESTS
                *
159      UPDT2  TEST L      XH*VSLAST,XH*PH5,ENDIT   CONTINUE IF UPDATES PENDING
160             SAVEVALUE   V$LAST+,1,XH     UPDATE TABLE POINTER LAST
161             ASSIGN      7,1,PH
                *
162      LOOPC  TEST L      PH7,PH6,FINUP    IF COUNTER=# OF BE GO TO FINUP
163             TEST L      PH7,PH5,ELSE
164             ASSIGN      15,PH7,PH        DETERMINE # OF BE TO BE UPDATED
165             TRANSFER    ,SKIP2
166      ELSE   ASSIGN      15,V$INCRE,PH
167      SKIP2  ASSIGN      9,V$BECNL,PH
168             ASSIGN      7+,1,PH          INCREMENT COUNTER
169             SPLIT       1,LOOPC          SEND OUT ANOTHER UPDATE TO NEXT BE
                *
170             QUEUE       PH5
171             ENTER       PH5
172             DEPART      PH5
173             SEIZE       PH5              SEIZE THE BE CPU
174             ADVANCE     PH13             BINT
175             RELEASE     PH5
176             LEAVE       PH5
177             ENTER       9                ENTER UPDATE PENDING STORAGE
178             QUEUE       PH9              QUEUE FOR CHANNEL BETWEEN 2 BE'S
179             SEIZE       PH9              BE TO BE CHANNEL
180             DEPART      PH9
181             ADVANCE     V$CHANL
182             ADVANCE     PH14             MESSAGE SYSTEM
183             RELEASE     PH9
184             QUEUE       PH15
185             ENTER       PH15
186             DEPART      PH15
187             ASSIGN      1,XH*VSLAST,PF
188             PRIORITY    3
189             SEIZE       PH15              SEIZE THE BE CPU
                *
                *       CHECK THE TIMESTAMPS OF SAME RECORD MODIFICATIONS AT THAT BE
                *
190      TIMCK  ASSIGN      5,XH*PH15,PF     INITIALIZE CNTR TO FINAL ENTRY IN UPDT TA
191             TEST NE     PF5,0,CONTU      IF THERE ARE NO ENTRIES,  CONTINUE.
192      LOOPD  TEST L      MX*PH5(PF1,3),MX*PH15(PF5,3),CONTU   IS TIMESTAMP LESS?
                *
                *       IF TIMESTAMP OLDER, DON'T PERFORM MODIFICATION
                *
193             TEST E      MX*PH5(PF1,2),MX*PH15(PF5,2),*+5   IS IT THE SAME RECORD?
194             RELEASE     PH15             RELEASE THE BACKEND PROCESSOR
195             LEAVE       PH15             LEAVE THE BE PARTITION
```

```
196             LEAVE     9                 LEAVE SECONDARY UPDATE STORAGE LIST
197             TRANSFER  ,ENDIT
198             ASSIGN    5-,1,PF           DECREMENT COUNTER
199             TEST E    PF5,0,LOOPD       LOOP IF MORE IN TABLE
200       CONTU ADVANCE   PH13              BINT ON NEXT BE
201             ASSIGN    4,2,PH            CODE UPDATE AS SECONDARY
202             ASSIGN    7,0,PH           COUNTER
203             ASSIGN    2,MX*PH5(PF1,1),PH   ASSIGN INTO PH2 THE # OF IO REQ
204             ASSIGN    5,PH15,PH        ASSIGN PH15 INTO PH5
205             TRANSFER  ,LOOP8
                START     800,,400
                NOXREF
                END

                                        **** ASSEMBLY TIME = .04 MINUTES ****
```

```
BLOCK
NUMBER  *LOC    OPERATION  A,B,C,D,E,F,G,H,I          COMMENTS
                SIMULATE
                RMULT      14523
                RMULT      92576
        *
        *         ELLIS MODEL
        *
        * **    ENTITY DEFINITIONS
        *
        *    PARAMETERS
        *
        *         PH1: TERMINAL ASSIGNMENT NUMBER
        *         PF1: HALFWORD SAVEVALUE WHOSE NUMBER IS 10+PH5
        *         PH2: NUMBER OF I/O REQUESTS NECESSARY FOR EACH USER RWQUEST
        *         PF2: MODIFICATION REQUEST NIMBER
        *         PH3: SIZE OF MESSAGE BUFFER
        *         PH4: MARKED 0 IF READ REQUEST
        *                     1 IF PRIMARY UPDATE REQUEST
        *                     2 IF SECONDARY UPDATE REQUEST
        *         PF4: MARKED 0 IF UPDATE REQUEST HAS NOT BEEN SPOOLED
        *                     1 IF UPDATE REQUEST HAS BEEN SPOOLED
        *         PH5: ID NUMBER OF THE BACKEND MACHINE BEING USED
        *         PH6: NUMBER OF BACKEND MACHINES USED IN THE SIMULATION
        *         PH7: A COUNTER
        *         PH8: NUMBER OF REQUESTS MADE BY THE USER
        *         PH9: CHANNEL ID USED BETWEEN TWO BACKEND MACHINES
        *         PF11: TRANSNISSION SPEED TO CHANNEL IN BITS/SEC
        *         PF12: CHANNEL TRANSMISSION SPEED IN BITS/SEC
        *         PH13: TIME USED BY HINT OR BINT
        *         PH14: TIME USED BY THE MESSAGE SYSTEM
        *         PH15: ID NUMBER OF THE BE THAT THE SECONDARY UPDATE IS SENT TO
        *
        *    FACILITIES
        *
        *         1 - 4:  CPU'S OF THE FOUR BACKEND MACHINES
        *         11,22,33,44:  CHANNELS FROM BACKEND TO DEVICE
        *         12->14: CHANNELS FROM BE1 TO BE2,BE3,AND BE4
        *         21,23,24: CHANNELS FROM BE2 TO BE1,BE3,AND BE4
        *         31,32,34: CHANNELS FROM BE3 TO BE1,BE2,AND BE4
        *         41->43: CHANNELS FROM BE4 TO BE1,BE2,AND BE3
        *         91->94:  CHANNELS BETWEEN HOST AND BE
        *         99: HOST CPU
        *         100: TERMINAL TO HOST CHANNEL
        *         101->109: TERMINALS CONNECTED TO HOST
        *
        *    HALFWORD SAVEVALUES
        *         XH1:  NUMBER OF MODIFICATION REQUSTS ACCEPTED BY HOST
        *
        *    DEFINITIONS
        *         UPDAT1: PRIMARY UPDATE
        *         UPDAT2: SECONDARY UPDATE
        *         LAST: ROW OF MATRIX LAST USED IN UPDATING
        *         CURRENT: NUMBER OF UPDATES SPOOLED TO THAT BE
        *
        *    STORAGES REPRESENT PARTITIONS IN EACH BACKEND
        *
```

```
         STORAGE    S1-S8,2
*
 TRMNL FVARIABLE  PH3*8/PF11*1000
 CHANL FVARIABLE  PH3*8/PF12*1000
 LAST  VARIABLE   10+PH5
 INCRE VARIABLE   PH7+1
 BECNL VARIABLE   10*PH5+PH15
 TYPE  FUNCTION   RN2,D2
.80,0/1.00,1
*        20% UPDATE
 TERM  FUNCTION   RN2,D8
.125,101/.25,102/.375,103/.5,104/.625,105/.75,106/.875,107/1.0,108
 REQNO FUNCTION   RN2,D10
.10,1/.20,2/.30,4/.40,5/.50,7/.60,9/.70,11/.80,12/.90,13/1.00,14
 LNGTH FUNCTION   RN2,D2
.90,128/1.00,256
*  90% FIT IN ONE BUFFER
 HCBE  FUNCTION   PH5,D8          HOST TO BE CHANNELS
1,91/2,92/3,93/4,94/5,95/6,96/7,97/8,98
 BEDEV FUNCTION   PH5,D8          BE TO DEVICE CHANNELS
1,11/2,22/3,33/4,44/5,55/6,66/7,77/8,88
 USREQ FUNCTION   RN2,D9          # OF REQUESTS BY A USER
.10,1/.20,3/.40,5/.50,7/.60,9/.70,11/.80,13/.90,15/1.00,17
*
 RNDM  FUNCTION   RN2,D4
.25,1/.50,2/.75,3/1.00,4
*
         INITIAL    XH1-XH4,0      INITIALIZE SAVEVALUES 1-10 TO ZERO
*
1        GENERATE   1500,50,,15,,15PF,15PH  GENERATE 15 TRANSX, AVE 1/1.5 SEC
2        PRIORITY   1
3        ASSIGN     1,FN$TERM,PH     ASSIGN TERMINAL # IN PH1
4        ASSIGN     6,4,PH           NUMBER OF BACKENDS IN SIMULATION
5        ASSIGN     8,FN$USREQ,PH    STORE # OF USER'S COMMANDS IN PH8
6        ASSIGN     11,50000,PF         TRANSMISSION SPEED TO CHANNEL
7        ASSIGN     12,50000,PF         TRANSMISSION SPEED OF CHANNEL
8        ASSIGN     13,5,PH          ADVANCE TIME FOR HINT OR BINT
9        ASSIGN     14,5,PH          ADVANCE TIME FOR MESSAGE SYSTEM
10       QUEUE      PH1              QUEUE FOR TERMINAL
11       SEIZE      PH1              SEIZE THE TERMINAL
12       DEPART     PH1              DEPART QUEUE FOR TERMINAL
*
*              MAIN LOOP FOR USER REQUESTS
*
13       MORE   ADVANCE    1000,500         TIME TAKEN TO TYPE REQUEST
14       ASSIGN     8-,1,PH          DECREMENT # OF REQUESTS MADE BY USER
15       QUEUE      100              QUEUE HOST/TERMINAL CHANNEL
16       SEIZE      100              TERMINAL-HOST CHANNEL
17       DEPART     100              LEAVE QUEUE HOST-TERMINAL CHANNEL
18       ADVANCE    V$TRMNL          LINE TRANSMISSION
19       RELEASE    100              RELEASE CHANNEL
20       QUEUE      99               QUEUE FOR CPU
21       SEIZE      99               SEIZE HOST CPU
22       DEPART     99
23       ASSIGN     2,FN$REQNO,PH    ASSIGN # OF IO REQUESTS IN PH 2
24       ASSIGN     3,FN$LNGTH,PH    LENGTH OF BUFFER TRANSMISSION
25       ASSIGN     4,FN$TYPE,PH     ASSIGN UPDATE OR READ
```

```
26              ADVANCE     PH13                HINT
27              ADVANCE     PH14                MESSAGE SYSTEM
28              TEST NE     PH4,1,UPDT          IF UPDATE GO TO UPDT FOR ASSIGNMENT
29              ASSIGN      5,1,PH
            *
            *       CHECK FOR A FREE MACHINE
            *
30          LOOPA RELEASE   99                  FREE HOST CPU
31              QUEUE       FNSHOBE             WAIT FOR HOST-BE CHANNEL
32              SEIZE       FNSHOBE             SEIZE HOST-BE CHANNEL
33              DEPART      FNSHOBE
34              ADVANCE     V$CHANL
35              ADVANCE     1                   TIME TAKEN TO CHECK IF FREE
36              GATE SNE    PH5,BEDB            GO TO BEDB IF BE FREE
37              RELEASE     FNSHOBE             RETURN CHANNEL
38              QUEUE       99
39              SEIZE       99                  SEIZE HOST CPU
40              DEPART      99
41              ASSIGN      5+,1,PH
42              TEST LE     PH5,PH6,FULL        SEND TO FULL IF ALL BE'S ARE BUSY
43              TRANSFER    ,LOOPA
44          FULL ASSIGN     5,FNSRNDM,PH        RANDOMLY ASSIGN BE MACHINE
45              ADVANCE     1                   ASSIGNMENT TIME
46              RELEASE     99                  RELEASE THE HOST PROCESSOR
            *
            *               BACKEND HAS BEEN SELECTED
            *
47          ENTER QUEUE     FNSHOBE
48              SEIZE       FNSHOBE
49              DEPART      FNSHOBE
50              ADVANCE     V$CHANL
51          BEDB RELEASE    FNSHOBE
52              ADVANCE     PH13                BINT
53              QUEUE       PH5                 QUEUE FOR BE PARTITION
54              ENTER       PH5                 ENTER BE PARTITION
55              DEPART      PH5
56              SEIZE       PH5                 SEIZE THE BE CPU
57              ASSIGN      7,0,PH              PH7 IS A COUNTER
            *
            *               PROCESSING OF IO REQUESTS
            *
58          LOOPB TEST NE   PH2,PH7,READ        IF IO DONE GO TO READ
59              TEST NE     PH4,1,*+2
60              RELEASE     PH5
61              QUEUE       FNSBEDEV
62              SEIZE       FNSBEDEV            SEIZE BE DEVICE FOR IO
63              DEPART      FNSBEDEV
64              ADVANCE     52                  TIME FOR EACH IO
65              TEST E      PH4,1,*+4           IS REQUEST A PRIMARY UPDATE?
66              TEST NE     PF4,1,*+3           HAS THIS UPDATE ALREADY BEEN SPOOLED?
67              ADVANCE     30                  IF NOT, SPOOL THE UPDATE
68              ASSIGN      4,1,PF              MARK THE UPDATE AS SPOOLED
69              RELEASE     FNSBEDEV            RELEASE DEVICE
70              TEST NE     PH4,1,*+2
71              SEIZE       PH5                 SEIZE THE BE CPU
72              ADVANCE     1                   PROCESSING TIME
73              ASSIGN      7+,1,PH             INCREMENT COUNTER
```

```
74              TRANSFER    ,LOOPB
                *
                *
                *               IO COMPLETED
                *
                *
                *               RETURN INFORMATION TO THE HOST AND TERMINALS
                *
75          READ  ADVANCE     PH13              BINT
76                RELEASE     PH5
77                LEAVE       PH5               LEAVE PARTITION IN BE
78                QUEUE       FN$HOBE           WAIT FOR HOST-BE CHANNEL
79                SEIZE       FN$HOBE
80                DEPART      FN$HOBE
81                ADVANCE     V$CHANL
82                ADVANCE     PH14              MESSAGE SYSTEM
83                RELEASE     FN$HOBE
84                TEST E      PH4,1,*+3
85                LOGICR      PF2               RESET THE LOGIC SWITCH TO FREE HELD TRANS
86                TRANSFER    ,ENDIT
87          RETRN QUEUE       99                QUEUE FOR HOST
88                PREEMPT     99,PR             PREEMPT HOST CPU
89                DEPART      99
90                ADVANCE     PH13              HINT
91                RETURN      99
92                QUEUE       100               WAIT FOT CPU-TERMINAL CHANNEL
93                PREEMPT     100,PR
94                DEPART      100
95                ADVANCE     V$TRMNL
96                RETURN      100
97                TEST E      PH8,0,MORE        MORE REQUESTS BY THIS USER?
98                RELEASE     PH1               RELEASE OCCUPIED TERMINAL
99          ENDIT TERMINATE   1
                *
                *
                *               ROUTINE CALLED FOR MODIFICATION REQUESTS
                *
100         UPDT  ASSIGN      7,1,PH
101               SAVEVALUE   1+,1,XH           INCREMENT XH1—THE UPDATE COUNTER
102               ASSIGN      2,XH1,PF          ASSIGN UPDATE # TO PF2
103               LOGICS      PF2               SET LOGIC SWITCH PF2
104         LOOPC TEST LE     PH7,PH6,WAIT      IF COUNTER=# OF BE GO TO WAIT
105               ASSIGN      5,PH7,PH          DETERMINE # OF BE TO BE UPDATED
106               ASSIGN      7+,1,PH           INCREMENT COUNTER
107               SPLIT       1,ENTER           SEND OUT TRANSX COPY TO BE UPDATED
108               TRANSFER    ,LOOPC            SEND THE PARENT TRANSX TO LOOPC
                *
109         WAIT  RELEASE     99                RELEASE THE HOST PROCESSOR
110               GATE LR     PF2               WAIT UNTIL LOGIC SWITCH PF2 IS RESET
111               TRANSFER    ,RETRN            GO TO RETRN WHEN THE MODIFICATION REQUEST
                *                               HAS BEEN COMPLETED BY ONE OF THE BACKENDS
                *
                  NOXREF
                  START       800,,100
                  END

                                          **** ASSEMBLY TIME = .03 MINUTES ****
```

```
BLOCK
NUMBER  *LOC    OPERATION   A,B,C,D,E,F,G,H,I          COMMENTS
                SIMULATE
                RMULT       92576
                RMULT       14523
*
*         COCASYL MODEL
*
* **    ENTITY DEFINITIONS
*
*    PARAMETERS
*
*         PH1: TERMINAL ASSIGNMENT NUMBER
*         PF1: HALFWORD SAVEVALUE WHOSE NUMBER IS 10+PH5
*         PH2: NUMBER OF I/O REQUESTS NECESSARY FOR EACH USER RWQUEST
*         PH3: SIZE OF MESSAGE BUFFER
*         PH4: MARKED 0 IF READ REQUEST
*                     1 IF PRIMARY UPDATE REQUEST
*                     2 IF SECONDARY UPDATE REQUEST
*         PF4: MARKED 0 IF UPDATE REQUEST HAS NOT BEEN SPOOLED
*                     1 IF UPDATE REQUEST HAS BEEN SPOOLED
*         PH5: ID NUMBER OF THE BACKEND MACHINE BEING USED
*         PH6: NUMBER OF BACKEND MACHINES USED IN THE SIMULATION
*         PH7: A COUNTER
*         PH8: NUMBER OF REQUESTS MADE BY THE USER
*         PH9: CHANNEL ID USED BETWEEN TWO BACKEND MACHINES
*         PF11: TRANSMISSION SPEED TO CHANNEL IN BITS/SEC
*         PF12: CHANNEL TRANSMISSION SPEED IN BITS/SEC
*         PH13: TIME USED BY HINT OR BINT
*         PH14: TIME USED BY THE MESSAGE SYSTEM
*         PF14:  LAST TRANSACTION MARKER
*         PH15: ID NUMBER OF THE BE THAT THE SECONDARY UPDATE IS SENT TO
*
*    FACILITIES
*
*         1 - 4:  CPU'S OF THE FOUR BACKEND MACHINES
*         11,22,33,44:  CHANNELS FROM BACKEND TO DEVICE
*         12->14: CHANNELS FROM BE1 TO BE2,BE3,AND BE4
*         21,23,24: CHANNELS FROM BE2 TO BE1,BE3,AND BE4
*         31,32,34: CHANNELS FROM BE3 TO BE1,BE2,AND BE4
*         41->43: CHANNELS FROM BE4 TO BE1,BE2,AND BE3
*         91->94:  CHANNELS BETWEEN HOST AND BE
*         99: HOST CPU
*         100: TERMINAL TO HOST CHANNEL
*         101->109: TERMINALS CONNECTED TO HOST
*
*    HALFWORD SAVEVALUES
*         XH1:  CURRENT TABLE POINTER FOR BACKEND 1  (PRIMARY BACKEND)
*         XH2:  LAST TABLE POINTER FOR PRIMARY BACKEND
*
*    DEFINITIONS
*         UPDAT1: PRIMARY UPDATE
*         UPDAT2: SECONDARY UPDATE
*         LAST: ROW OF MATRIX LAST USED IN UPDATING
*         CURRENT: NUMBER OF UPDATES SPOOLED TO THAT BE
*
*
```

```
*    HALFWORD MATRIX
*         1:  MODIFICATION TABLE FOR PRIMARY BACKEND
*
 1      MATRIX      MH,200,1
*
*    STORAGES REPRESENT PARTITIONS IN EACH BACKEND
*
        STORAGE     S1-S8,2
 TRMNL  FVARIABLE   PH3*8/PF11*1000
 CHANL  FVARIABLE   PH3*8/PF12*1000
 LAST   VARIABLE    10+PH5
 INCRE  VARIABLE    PH7+1
 BECNL  VARIABLE    10*PH5+PH15
 TYPE   FUNCTION    RN2,D2
.80,0/1.00,1
*   20% UPDATE
 TERM   FUNCTION    RN2,D8
.125,101/.25,102/.375,103/.5,104/.625,105/.75,106/.875,107/1.0,108
 REQNO  FUNCTION    RN2,D10
.10,1/.20,2/.30,4/.40,5/.50,7/.60,9/.70,11/.80,12/.90,13/1.00,14
 LNGTH  FUNCTION    RN2,D2
.90,128/1.00,256
*   90% FIT IN ONE BUFFER
 HOBE   FUNCTION    PH5,D8              HOST TO BE CHANNELS
1,91/2,92/3,93/4,94/5,95/6,96/7,97/8,98
 BEDEV  FUNCTION    PH5,D8              BE TO DEVICE CHANNELS
1,11/2,22/3,33/4,44/5,55/6,66/7,77/8,88
 USREQ  FUNCTION    RN2,D9              # OF REQUESTS BY A USER
.10,1/.20,3/.40,5/.50,7/.60,9/.70,11/.80,13/.90,15/1.00,17
 RNDM   FUNCTION    RN2,D4
.25,1/.50,2/.75,3/1.00,4
        INITIAL     XH1-XH9,0         INITIALIZE SAVEVALUES 1-9 TO ZERO
        INITIAL     XH11-XH18,0       INITIALIZE SAVEVALUES 11-18 TO ZERO
*
1       GENERATE    1500,50,,15,,15PF,15PH  GENERATE 15 TRANSX, AVE 1/1.5 SEC
2       PRIORITY    1
3       ASSIGN      1,FNSTERM,PH        ASSIGN TERMINAL # IN PH1
4       ASSIGN      6,4,PH              NUMBER OF BACKENDS IN SIMULATION
5       ASSIGN      8,FNSUSREQ,PH       STORE # OF USER'S COMMANDS IN PH8
6       ASSIGN      11,50000,PF         TRANSMISSION SPEED TO CHANNEL
7       ASSIGN      12,50000,PF         TRANSMISSION SPEED OF CHANNEL
8       ASSIGN      13,5,PH             ADVANCE TIME FOR HINT OR BINT
9       ASSIGN      14,5,PH             ADVANCE TIME FOR MESSAGE SYSTEM
10      QUEUE       PH1                 QUEUE FOR TERMINAL
11      SEIZE       PH1                 SEIZE THE TERMINAL
12      SAVEVALUE   5+,1,XH
13      TEST E      XH5,15,*+2          IF THIS IS THE LAST TRANSX, MARK IT
14      ASSIGN      14,1,PF             MARK AS THE LAST TRANSACTION
15      DEPART      PH1                 DEPART QUEUE FOR TERMINAL
*
*              MAIN LOOP FOR USER REQUESTS
*
16     MORE ADVANCE     1000,500          TIME TAKEN TO TYPE REQUEST
17      ASSIGN      8-,1,PH             DECREMENT # OF REQUESTS MADE BY USER
18      QUEUE       100                 QUEUE HOST/TERMINAL CHANNEL
19      SEIZE       100                 TERMINAL-HOST CHANNEL
20      DEPART      100                 LEAVE QUEUE HOST-TERMINAL CHANNEL
```

```
21              ADVANCE     VSTRMNL             LINE TRANSMISSION
22              ADVANCE     VSCHANL             CHANNEL TRANSMISSION
23              RELEASE     100                 RELEASE CHANNEL
24              QUEUE       99                  QUEUE FOR CPU
25              SEIZE       99                  SEIZE HOST CPU
26              DEPART      99
27              ASSIGN      2,FNSREQNO,PH       ASSIGN # OF IO REQUESTS IN PH 2
28              ASSIGN      3,FNSLNGTH,PH       LENGTH OF BUFFER TRANSMISSION
29              ASSIGN      4,FNSTYPE,PH        ASSIGN UPDATE OR READ
30              ADVANCE     PH13                HINT
31              ADVANCE     PH14                MESSAGE SYSTEM
32              ASSIGN      5,1,PH
33              TEST NE     PH4,1,ASIGN         IF UPDATE GO TO ASIGN FOR ASSIGNMENT
                *
                *       CHECK FOR A FREE MACHINE
                *
34        LOOPA RELEASE     99                  FREE HOST CPU
35              QUEUE       FN$HOBE             WAIT FOR HOST-BE CHANNEL
36              SEIZE       FN$HOBE             SEIZE HOST-BE CHANNEL
37              DEPART      FN$HOBE
38              ADVANCE     VSCHANL
39              ADVANCE     1                   TIME TAKEN TO CHECK IF FREE
40              GATE SNE    PH5,BEDB            GO TO BEDB IF BE FREE
41              RELEASE     FN$HOBE             RETURN CHANNEL
42              QUEUE       99
43              SEIZE       99                  SEIZE HOST CPU
44              DEPART      99
45              ASSIGN      5+,1,PH
46              TEST LE     PH5,PH6,FULL        SEND TO FULL IF ALL BE'S ARE BUSY
47              TRANSFER    ,LOOPA
48        FULL  ASSIGN      5,FNSRNDM,PH        RANDOMLY ASSIGN BE MACHINE
49        ASIGN ADVANCE     1                   ASSIGNMENT TIME
50              RELEASE     99                  RELEASE THE HOST PROCESSOR
                *
                *             BACKEND HAS BEEN SELECTED
                *
51              QUEUE       FN$HOBE
52              SEIZE       FN$HOBE
53              DEPART      FN$HOBE
54              ADVANCE     VSCHANL
55        BEDB  RELEASE     FN$HOBE
56              ADVANCE     PH13                BINT
57              QUEUE       PH5                 QUEUE FOR BE PARTITION
58              ENTER       PH5                 ENTER BE PARTITION
59              DEPART      PH5
60              TEST E      PH4,1,*+2
61              PRIORITY    3
62              SEIZE       PH5
63              ASSIGN      7,0,PH              PH7 IS A COUNTER
                *
                *             PROCESSING OF IO REQUESTS
                *
64        LOOPB TEST NE     PH2,PH7,FINIO       IF IO DONE GO TO FINIO
65              TEST NE     PH4,1,*+2
66              RELEASE     PH5
67              QUEUE       FN$BEDEV
68              SEIZE       FN$BEDEV            SEIZE BE DEVICE FOR IO
```

```
69              DEPART      FN$8EDEV
70              ADVANCE     52                  TIME FOR EACH IO
71              TEST E      PH4,1,*+4           IS REQUEST A PRIMARY UPDATE?
72              TEST NE     PF4,1,*+3           HAS THIS UPDATE ALREADY BEEN SPOOLED?
73              ADVANCE     30                  IF NOT, SPOOL THE UPDATE
74              ASSIGN      4,1,PF              MARK THE UPDATE AS SPOOLED
75              RELEASE     FN$8EDEV            RELEASE DEVICE
76              TEST NE     PH4,1,*+2
77              SEIZE       PH5
78              ADVANCE     1                   PROCESSING TIME
79              ASSIGN      7+,1,PH             INCREMENT COUNTER
80              TRANSFER    ,LOCP8
          *
          *               IO COMPLETED
          *
          *
          *         ENTER MODIFICATION REQUESTS IN THE MODIFICATION TABLE
          *
81         FINIO TEST NE    PH4,1,UPOT1         IF UPDATE, GO TO UPOT1-CHANGE TABLE
82               TEST E     PH4,2,READ          IF QUERY ONLY COMMAND, GO TO READ
          *
          *         REQUEST IS A SECONDARY UPDATE
          *
83          SNDUP RELEASE    PH5
84               LEAVE      PH5                 RELEASE THE BE PARTITION
85          FINUP TEST G     S*PH5,0,UPOT2       IF BE PARTITION EMPTY GO TO UPDATES
86               TRANSFER   ,ENDIT              ELSE END TRANSACTION
          *
          *
          *               RETURN INFORMATION TO THE HOST AND TERMINALS
          *
87          READ  ADVANCE    PH13                HINT
88               RELEASE    PH5
89               LEAVE      PH5                 LEAVE PARTITION IN BE
90               QUEUE      FN$HOBE             WAIT FOR HOST-BE CHANNEL
91               SEIZE      FN$HOBE
92               DEPART     FN$HOBE
93               ADVANCE    VSCHANL
94               ADVANCE    PH14                MESSAGE SYSTEM
95               RELEASE    FN$HOBE
96               QUEUE      99                  QUEUE FOR HOST
97               PREEMPT    99,PR               PREEMPT HOST CPU
98               DEPART     99
99               ADVANCE    PH13                HINT
100              RETURN     99
101              QUEUE      100                 WAIT FOT CPU-TERMINAL CHANNEL
102              PREEMPT    100,PR
103              DEPART     100
104              ADVANCE    VSTRMNL
105              RETURN     100
106              TEST E     PH8,0,MORE          MORE REQUESTS BY THIS USER?
107              RELEASE    PH1                 RELEASE OCCUPIED TERMINAL
          *
          *         IF LAST USER, BEGIN SECONDARY UPDATES
          *
108              TEST E     PF14,1,*+2          IF LAST TRANSACTION  SPLIT
109              SPLIT      1,UPT2
```

```
110      ENDIT TERMINATE  1                                          84
         *
         *        AOC REQUEST TO MODIFICATION TABLE
         *
111      UPDT1 SAVEVALUE  PH5+,1,XH        INCREMENT # OF UPDATES SPOOLED
112      ONE   MSAVEVALUE 1,XH1,1,PH2,MH   STORE THE NUMBER OF IO REQUESTS
113      SKIP1 ADVANCE    1                 TIME TAKEN TO WRITE TO TABLE
114            TRANSFER   ,READ
         *
         *              ROUTINE CALLED FOR SECONDARY MODIFICATION REQUESTS
         *
115      UPT2  SAVEVALUE  2+,1,XH
116            ASSIGN     14,0,PF
117      UPDT2 ASSIGN     5,1,PH
118            TEST L     XH2,XH*PH5,ENDIT   CONTINUE IF UPDATES PENDING
119            ASSIGN     1,XH2,PF          ASSIGN UPDATE TABLE ROW # INTO PF1
120            SAVEVALUE  2+,1,XH           INCREMENT UPDT LIST COUNTER
121            ASSIGN     7,1,PH
         *
         *        SEND THE SECONDARY MODIFICATIONS
         *
122      LOOPC TEST LE    PH7,PH6,UPDT2     IF COUNTER GT # OF BE GO TO FINUP
123            TEST NE    PH7,PH5,ELSE      GO TO ELSE IF CNTR = # OF BE
124            ASSIGN     15,PH7,PH         DETERMINE # OF BE TO BE UPDATED
125            TRANSFER   ,SKIP2
126      ELSE  ASSIGN     7+,1,PH           INCREMENT COUNTER
127            TRANSFER   ,LOOPC
128      SKIP2 ASSIGN     9,V$BECNL,PH
129            ASSIGN     7+,1,PH           INCREMENT COUNTER
130            SPLIT      1,LOOPC           SEND OUT ANOTHER UPDATE TO NEXT BE
131            ADVANCE    2000              2000 MSEC PAUSE BETWEEN SENDING OF UPDATE
         *
132            QUEUE      PH5
133            ENTER      PH5
134            DEPART     PH5
135            SEIZE      PH5
136            ADVANCE    PH13              BINT
137            RELEASE    PH5
138            LEAVE      PH5
139            QUEUE      PH9               QUEUE FOR CHANNEL BETWEEN 2 BE'S
140            SEIZE      PH9               BE TO BE CHANNEL
141            DEPART     PH9
142            ADVANCE    V$CHANL
143            ADVANCE    PH14              MESSAGE SYSTEM
144            RELEASE    PH9
145            QUEUE      PH15
146            ENTER      PH15
147            DEPART     PH15
148            SEIZE      PH15
149            ADVANCE    PH13              BINT ON NEXT BE
150            ASSIGN     4,2,PH            CODE UPDATE AS SECONDARY
151            ASSIGN     7,0,PH            COUNTER
152            ASSIGN     2,MH*PH5(PF1,1),PH   ASSIGN INTO PH2 THE # OF IO REQ
153            ASSIGN     5,PH15,PH         ASSIGN PH15 INTO PH5
154            TRANSFER   ,LOOPB
               START      800,,400
               NOXREF
```

```
BLOCK
NUMBER  *LOC    OPERATION   A,B,C,D,E,F,G,H,I          COMMENTS
                SIMULATE
                RMULT       92576
                RMULT       14523
*
*         HYBRID MODEL
*
* **     ENTITY DEFINITIONS
*
*    PARAMETERS
*
*         PH1: TERMINAL ASSIGNMENT NUMBER
*         PF1: HALFWORD SAVEVALUE WHOSE NUMBER IS 10+PH5
*         PH2: NUMBER OF I/O REQUESTS NECESSARY FOR EACH USER RWQUEST
*         PH3: SIZE OF MESSAGE BUFFER
*         PH4: MARKED 0  IF READ REQUEST
*                     1  IF PRIMARY UPDATE REQUEST
*                     2  IF SECONDARY UPDATE REQUEST
*         PF4: MARKED 0  IF UPDATE REQUEST HAS NOT BEEN SPOOLED
*                     1  IF UPDATE REQUEST HAS BEEN SPOOLED
*         PH5: ID NUMBER OF THE BACKEND MACHINE BEING USED
*         PH6: NUMBER OF BACKEND MACHINES USED IN THE SIMULATION
*         PH7: A COUNTER
*         PH8: NUMBER OF REQUESTS MADE BY THE USER
*         PH9: CHANNEL ID USED BETWEEN TWO BACKEND MACHINES
*         PF11: TRANSNISSION SPEED TO CHANNEL IN BITS/SEC
*         PF12: CHANNEL TRANSMISSION SPEED IN BITS/SEC
*         PH13: TIME USED BY HINT OR BINT
*         PH14: TIME USED BY THE MESSAGE SYSTEM
*         PH15: ID NUMBER OF THE BE THAT THE SECONDARY UPDATE IS SENT TO
*
*    FACILITIES
*
*         1 - 4:  CPU'S OF THE FOUR BACKEND MACHINES
*         11,22,33,44:  CHANNELS FROM BACKEND TO DEVICE
*         12->14: CHANNELS FROM BE1 TO BE2,BE3,AND BE4
*         21,23,24: CHANNELS FROM BE2 TO BE1,BE3,AND BE4
*         31,32,34: CHANNELS FROM BE3 TO BE1,BE2,AND BE4
*         41->43: CHANNELS FROM BE4 TO BE1,BE2,AND BE3
*         91->94:  CHANNELS BETWEEN HOST AND BE
*         99: HOST CPU
*         100: TERMINAL TO HOST CHANNEL
*         101->109: TERMINALS CONNECTED TO HOST
*
*    HALFWORD SAVEVALUES
*         XH1->XH8: CURRENT TABLE POINTERS FOR BE1->BE8
*         XH11->XH18: LAST TABLE POINTERS FOR BE1->BE8
*
*    DEFINITIONS
*         UPDAT1: PRIMARY UPDATE
*         UPDAT2: SECONDARY UPDATE
*         LAST: ROW OF MATRIX LAST USED IN UPDATING
*         CURRENT: NUMBER OF UPDATES SPOOLED TO THAT BE
*
*
*    HALFWORD MATRIX
```

```
*         1: UPDATE TABLE FOR BE1
*         2: UPDATE TABLE FOR BE2
*         3: UPDATE TABLE FOR BE3
*         4: UPDATE TABLE FOR BE4
*         5: UPDATE TABLE FOR BE5
*         6: UPDATE TABLE FOR BE6
*         7: UPDATE TABLE FOR BE7
*         8: UPDATE TABLE FOR BE8
*
 1      MATRIX      MH,100,1
 2      MATRIX      MH,100,1
 3      MATRIX      MH,100,1
 4      MATRIX      MH,100,1
 5      MATRIX      MH,100,1
 6      MATRIX      MH,100,1
 7      MATRIX      MH,100,1
 8      MATRIX      MH,100,1
*
*    STORAGES REPRESENT PARTITIONS IN EACH BACKEND
*
        STORAGE     S1-S8,2
 TRMNL  FVARIABLE   PH3*8/PF11*1000
 CHANL  FVARIABLE   PH3*8/PF12*1000
 LAST   VARIABLE    10+PH5
 INCRE  VARIABLE    PH7+1
 BECNL  VARIABLE    10*PH5+PH15
 TYPE   FUNCTION    RN2,D2
.80,0/1.00,1
*   20% UPDATE
 TERM   FUNCTION    RN2,D8
.125,101/.25,102/.375,103/.5,104/.625,105/.75,106/.875,107/1.0,108
 REQNO  FUNCTION    RN2,D10
.10,1/.20,2/.30,4/.40,5/.50,7/.60,9/.70,11/.80,12/.90,13/1.00,14
 LNGTH  FUNCTION    RN2,D2
.90,128/1.00,256
*   90% FIT IN ONE BUFFER
 HOBE   FUNCTION    PH5,D8              HOST TO BE CHANNELS
1,91/2,92/3,93/4,94/5,95/6,96/7,97/8,98
 BEDEV  FUNCTION    PH5,D8              BE TO DEVICE CHANNELS
1,11/2,22/3,33/4,44/5,55/6,66/7,77/8,88
 USREQ  FUNCTION    RN2,D9              # OF REQUESTS BY A USER
.10,1/.20,3/.40,5/.50,7/.60,9/.70,11/.80,13/.90,15/1.00,17
 RNDM   FUNCTION    RN2,D4
.25,1/.50,2/.75,3/1.00,4
        INITIAL     XH1-XH8,0         INITIALIZE SAVEVALUES 1-10 TO ZERO
        INITIAL     XH11-XH18,0       INITIALIZE SAVEVALUES 11-18 TO ZERO
*
1       GENERATE    1500,50,,15,,15PF,15PH  GENERATE 15 TRANSX, AVE 1/1.5 SEC
2       PRIORITY    1
3       ASSIGN      1,FN$TERM,PH      ASSIGN TERMINAL # IN PH1
4       ASSIGN      6,4,PH            NUMBER OF BACKENDS IN SIMULATION
5       ASSIGN      8,FN$USREQ,PH     STORE # OF USER'S COMMANDS IN PH8
6       ASSIGN      11,50000,PF       TRANSMISSION SPEED TO CHANNEL
7       ASSIGN      12,50000,PF       TRANSMISSION SPEED OF CHANNEL
8       ASSIGN      13,5,PH           ADVANCE TIME FOR HINT OR BINT
9       ASSIGN      14,5,PH           ADVANCE TIME FOR MESSAGE SYSTEM
10      QUEUE       PH1               QUEUE FOR TERMINAL
```

```
11              SEIZE       PH1                 SEIZE THE TERMINAL
12              DEPART      PH1                 DEPART QUEUE FOR TERMINAL
                *
                *
                *               MAIN LOOP FOR USER REQUESTS
                *
13          MORE ADVANCE    1000,500            TIME TAKEN TO TYPE REQUEST
14               ASSIGN     8-,1,PH             DECREMENT # OF REQUESTS MADE BY USER
15               QUEUE      100                 QUEUE HOST/TERMINAL CHANNEL
16               SEIZE      100                 TERMINAL-HOST CHANNEL
17               DEPART     100                 LEAVE QUEUE HOST-TERMINAL CHANNEL
18               ADVANCE    VSTRMNL             LINE TRANSMISSION
19               ADVANCE    VSCHANL             CHANNEL TRANSMISSION
20               RELEASE    100                 RELEASE CHANNEL
21               QUEUE      99                  QUEUE FOR CPU
22               SEIZE      99                  SEIZE HOST CPU
23               DEPART     99
24               ASSIGN     2,FN$REQNO,PH       ASSIGN # OF IO REQUESTS IN PH 2
25               ASSIGN     3,FN$LNGTH,PH       LENGTH OF BUFFER TRANSMISSION
26               ASSIGN     4,FN$TYPE,PH        ASSIGN UPDATE OR READ
27               ADVANCE    PH13                HINT
28               ADVANCE    PH14                MESSAGE SYSTEM
29               ASSIGN     5,1,PH
30               TEST NE    PH4,1,FULL          IF UPDATE GO TO FULL FOR ASSIGNMENT
                *
                *               CHECK FOR A FREE MACHINE
                *
31          LOOPA RELEASE   99                  FREE HOST CPU
32               QUEUE      FN$HOBE             WAIT FOR HOST-BE CHANNEL
33               SEIZE      FN$HOBE             SEIZE HOST-BE CHANNEL
34               DEPART     FN$HOBE
35               ADVANCE    VSCHANL
36               ADVANCE    1                   TIME TAKEN TO CHECK IF FREE
37               GATE SNE   PH5,BEDB            GO TO BEDB IF BE FREE
38               RELEASE    FN$HOBE             RETURN CHANNEL
39               QUEUE      99
40               SEIZE      99                  SEIZE HOST CPU
41               DEPART     99
42               ASSIGN     5+,1,PH
43               TEST LE    PH5,PH6,FULL        SEND TO FULL IF ALL BE'S ARE BUSY
44               TRANSFER   ,LOOPA
45          FULL ASSIGN     5,FN$RNDM,PH        RANDOMLY ASSIGN BE MACHINE
46          ASIGN ADVANCE   1                   ASSIGNMENT TIME
47               RELEASE    99                  RELEASE THE HOST PROCESSOR
                *
                *               BACKEND HAS BEEN SELECTED
                *
48               QUEUE      FN$HOBE
49               SEIZE      FN$HOBE
50               DEPART     FN$HOBE
51               ADVANCE    VSCHANL
52          BEDB RELEASE    FN$HOBE
53               ADVANCE    PH13                BINT
54               QUEUE      PH5                 QUEUE FOR BE PARTITION
55               ENTER      PH5                 ENTER BE PARTITION
56               DEPART     PH5
57               SEIZE      PH5                 SEIZE THE BE CPU
```

```
58              ASSIGN      7,0,PH              PH7 IS A COUNTER
                *
                *               PROCESSING OF IO REQUESTS
                *
59          LOOPB TEST NE     PH2,PH7,FINIO     IF IO DONE GO TO FINIO
60                TEST NE     PH4,1,*+2
61.               RELEASE     PH5
62                QUEUE       FN$BEDEV
63                SEIZE       FN$BEDEV          SEIZE BE DEVICE FOR IO
64                DEPART      FN$BEDEV
65                ADVANCE     52                TIME FOR EACH IO
66                TEST E      PH4,1,*+4         IS REQUEST A PRIMARY UPDATE?
67                TEST NE     PF4,1,*+3         HAS THIS UPDATE ALREADY BEEN SPOOLED?
68                ADVANCE     30                IF NOT, SPOOL THE UPDATE
69                ASSIGN      4,1,PF            MARK THE UPDATE AS SPOOLED
70                RELEASE     FN$BEDEV          RELEASE DEVICE
71                TEST NE     PH4,1,*+2
72                SEIZE       PH5               SEIZE THE BE CPU
73                ADVANCE     1                 PROCESSING TIME
74                ASSIGN      7+,1,PH           INCREMENT COUNTER
75                TRANSFER    ,LOOPB
                *
                *               IO COMPLETED
                *
                *
                *               ENTER MODIFICATION REQUESTS IN MOD. TABLES
                *
76          FINIO TEST NE     PH4,1,UPDT1       IF UPDATE, GO TO UPDT1-CHANGE TABLE
77                TEST E      PH4,2,READ        IF QUERY ONLY COMMAND, GO TO READ
                *
                *               REQUEST IS A SECONDARY UPDATE
                *
78          RECPU RELEASE     PH5
79                LEAVE       PH5               RELEASE THE BE PARTITION
80          FINUP TEST G      S*PH5,1,UPDT2     IF BE PARTITION EMPTY GO TO UPDATES
81                TRANSFER    ,ENDIT            ELSE END TRANSACTION
                *
                *               RETURN INFORMATION TO THE HOST AND TERMINALS
                *
82          READ  ADVANCE     PH13              BINT
83                RELEASE     PH5
84                LEAVE       PH5               LEAVE PARTITION IN BE
85                TEST E      S*PH5,0,*+2       IF PARTITIONS EMPTY, SPLIT
86                SPLIT       1,UPDT2           SPLIT AND SEND 1 TO UPDT2
87                QUEUE       FN$HOBE           WAIT FOR HOST-BE CHANNEL
88                SEIZE       FN$HOBE
89                DEPART      FN$HOBE
90                ADVANCE     VSCHANL
91                ADVANCE     PH14              MESSAGE SYSTEM
92                RELEASE     FN$HOBE
93                QUEUE       99                QUEUE FOR HOST
94                PREEMPT     99,PR             PREEMPT HOST CPU
95                DEPART      99
96                ADVANCE     PH13              HINT
97                RETURN      99
98                QUEUE       100               WAIT FOT CPU-TERMINAL CHANNEL
99                PREEMPT     100,PR
```

```
100             DEPART      100
101             ADVANCE     V$TRMNL
102             RETURN      100
103             TEST E      PH8,0,MORE          MORE REQUESTS BY THIS USER?
104             RELEASE     PH1                 RELEASE OCCUPIED TERMINAL
105     ENDIT   TERMINATE   1
        *
        *               ADD REQUEST TO MODIFICATION TABLE
        *
106     UPDT1   SAVEVALUE   PH5+,1,XH           INCREMENT # OF UPDATES SPOOLED
107             TEST NE     PH5,1,ONE           DETERMINE BE YOU ARE UPDATING ON
108             TEST NE     PH5,2,TWO           SO YOU CAN UPDATE APPROPRIATE TABLE.
109             TEST NE     PH5,3,THREE
110             TEST NE     PH5,4,FOUR
111             TEST NE     PH5,5,FIVE
112             TEST NE     PH5,6,SIX
113             TEST NE     PH5,7,SEVEN
114             TEST NE     PH5,8,EIGHT
115     EIGHT   MSAVEVALUE  8,XH8,1,PH2,MH      STORE THE NUMBER OF IO REQUESTS
116             TRANSFER    ,SKIP1
117     SEVEN   MSAVEVALUE  7,XH7,1,PH2,MH      STORE THE NUMBER OF IO REQUESTS
118             TRANSFER    ,SKIP1
119     SIX     MSAVEVALUE  6,XH6,1,PH2,MH      STORE THE NUMBER OF IO REQUESTS
120             TRANSFER    ,SKIP1
121     FIVE    MSAVEVALUE  5,XH5,1,PH2,MH      STORE THE NUMBER OF IO REQUEXTS
122             TRANSFER    ,SKIP1
123     FOUR    MSAVEVALUE  4,XH4,1,PH2,MH      STORE THE NUMBER OF IO REQUEXTS
124             TRANSFER    ,SKIP1
125     THREE   MSAVEVALUE  3,XH3,1,PH2,MH      STORE THE NUMBER OF IO REQUESTS
126             TRANSFER    ,SKIP1
127     TWO     MSAVEVALUE  2,XH2,1,PH2,MH      STORE THE NUMBER OF IO REQUESTS
128             TRANSFER    ,SKIP1
129     ONE     MSAVEVALUE  1,XH1,1,PH2,MH      STORE THE NUMBER OF IO REQUESTS
130             TRANSFER    ,SKIP1
131     SKIP1   ADVANCE     1                   TIME TAKEN TO WRITE TO TABLE
132             TRANSFER    ,READ
        *
        *               ROUTINE CALLED FOR SECONDARY MODIFICATION REQUESTS
        *
133     UPDT2   TEST L      XH*V$LAST,XH*PH5,ENDIT   CONTINUE IF UPDATES PENDING
134             SAVEVALUE   V$LAST+,1,XH        UPDATE TABLE POINTER LAST
135             ASSIGN      7,1,PH
        *
        *               SEND THE SECONDARY MODIFICATIONS
        *
136     LOOPC   TEST LE     PH7,PH6,FINUP       IF COUNTER GT # OF BE GO TO FINUP
137             TEST NE     PH7,PH5,ELSE
138             ASSIGN      15,PH7,PH           DETERMINE # OF BE TO BE UPDATED
139             TRANSFER    ,SKIP2
140     ELSE    ASSIGN      7+,1,PH
141             TRANSFER    ,LOOPC
142     SKIP2   ASSIGN      9,V$BECNL,PH
143             ASSIGN      7+,1,PH             INCREMENT COUNTER
144             SPLIT       1,LOOPC             SEND OUT ANOTHER UPDATE TO NEXT BE
        *
145             QUEUE       PH5
146             ENTER       PH5
```

```
147             DEPART      PH5
148             SEIZE       PH5
149             ADVANCE     PH13            BINT
150             RELEASE     PH5
151             LEAVE       PH5
152             QUEUE       PH9             QUEUE FOR CHANNEL BETWEEN 2 BE'S
153             SEIZE       PH9              BE TO BE CHANNEL
154             DEPART      PH9
155             ADVANCE     V$CHANL
156             ADVANCE     PH14             MESSAGE SYSTEM
157             RELEASE     PH9
158             QUEUE       PH15
159             ENTER       PH15
160             DEPART      PH15
161             SEIZE       PH15
162             ADVANCE     PH13             BINT ON NEXT BE
163             ASSIGN      4,2,PH           CODE UPDATE AS SECONDARY
164             ASSIGN      7,0,PH          COUNTER
165             ASSIGN      1,XH*V$LAST,PF
166             ASSIGN      2,MH*PH5(PF1,1),PH   ASSIGN INTO PH2 THE # OF IO REQ
167             ASSIGN      5,PH15,PH       ASSIGN PH15 INTO PH5
168             TRANSFER    ,LOOP8
                *
                START       800,,400
                NOXREF
                END

                                        **** ASSEMBLY TIME = .04 MINUTES ****
```

A SIMULATION STUDY COMPARING FIVE CONSISTENCY ALGORITHMS FOR REDUNDANT DATABASES

by

KEVIN E. NORSWORTHY

B.G.S. University of Kansas, Lawrence, 1977

AN ABSTRACT OF

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

Kansas State University

Manhattan, Kansas

1978

ABSTRACT

This report describes a simulation evaluating the lative performance of five algorithms designed to keep dundant databases consistent. The throughput of each of e five algorithms is measured in an environment designed saturate system resources.

The performance of the models with different job mixes d the use of additional processors to increase throughput considered.

Simulation models of the following algorithms are esented:

1. Johnson and Thomas model
2. Ellis model
3. Codasyl model
4. Bernstein model
5. Hybrid model

The throughput of each of these models is measured in veral environments and compared to each other. The vantages and disadvantages of each model are discussed in rms of performance, simplicity, and proven ability to intain consistency.