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

1.0 Introduction.........................................................1
   1.1 Overview.......................................................1
   1.2 Background....................................................2

2.0 The Models..........................................................7
   2.1 Overview.......................................................7
   2.2 Codasyl Model................................................8
   2.3 Hybrid Model................................................9
   2.4 Ellis Model...................................................10
   2.5 Bernstein Model............................................11
   2.6 Johnson and Thomas Model................................12

3.0 Simulation Implementation.......................................15
   3.1 Overview.......................................................15
   3.2 Model Description..........................................15
   3.3 Explanation of the Bernstein Model........................17
   3.4 Actions Taken in Processing of Modification Requests...18
   3.5 Modifications Necessary for the Other Models............20

4.0 Results and Analysis.............................................30
   4.1 Overview.......................................................30
   4.2 Global Trends...............................................30
   4.3 Analysis of Change Within a Model........................52
   4.4 Analysis of Between Model Performance....................52
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

1.1 Multiprocessor Backend Configuration.........................5
3.1 Read-only Request Processing for Bernstein Model...........19
3.2 Modification Request Processing for Bernstein Model........21
3.3 Changes Needed for Johnson and Thomas Model...............23
3.4 Changes Needed for Ellis Model............................24
3.5 Changes Needed for Godasyl Model...........................26
3.6 Changes Needed for Hybrid Model............................27
3.7 Changes Needed for Hybrid II Model..........................29
4.1 Two Backends, 20% Modifications Job Mix....................34
4.2 Two Backends, 35% Modifications Job Mix....................36
4.3 Two Backends, 50% Modifications Job Mix....................37
4.4 Four Backends, 20% Modifications Job Mix...................38
4.5 Four Backends, 35% Modifications Job Mix...................39
4.6 Four Backends, 50% Modifications Job Mix...................41
4.7 Eight Backends, 20% Modifications Job Mix..................42
4.8 Eight Backends, 35% Modifications Job Mix..................43
4.9 Eight Backends, 50% Modifications Job Mix..................44
4.10 Communication Delay Comparison For Two Backends............49
4.11 Communication Delay Comparison For Four Backends...........50
4.12 Communication Delay Comparison For Eight Backends.........51
4.13 Johnson and Thomas Modification Anomaly....................54
4.14 Hybrid to Hybrid II Model Comparsion.......................56
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 algorithms. This study was undertaken to accomplish such a comparison by coding five of the algorithms in GESS-V and comparing their performance as measured by the amount of time taken to completely process several sample request streams.

1.2 BACKGROUND

Distributed database management systems have evolved as organizations found the need to simultaneously expand their data processing power and increase their data accessibility. An economical means of satisfying that need can be realized by use of a distributed database 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 functions. The advantages of a backend processor are summarized below.

1) It may free the resources of the host machine.
2) It reduces software overhead on the host.
3) It provides additional concurrency in the system.
4) It permits additional applications to be run on the host thus increasing throughput.
5) It can provide additional security since the privilege of executing an application program 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 database 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 eliminating 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
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 accessibility. 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 CCIASYL 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 inconsistency after all of the modifications saved by the primary database have been completed on the secondary databases, and the currency of the information stored in the secondary databases. When processing is light on the primary database, all of the secondary databases will contain information that is nearly current. The major disadvantage of this model is the possibility of overloading the primary backend with modification requests.

2.4 EIIIS MCDMEL

Ellis (3) presented a decentralized approach to the management of redundant databases. With his model modification requests may originate at any node in the network. The node of origin broadcasts a request to modify the record specified to all other nodes in the system. The node of 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 modification and sends the modification to the other nodes in the network. If the node of origin receives a negative acknowledgement from any of the other nodes, it suspends activity on the modification and resubmits it again at a later date. Positive acknowledgement is given if the secondary 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 ASS. 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
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
Modification Processing
For The Bernstein Model
Start

Seize host processor

Test: Is request a Modification? YES Timestamp modification Request

NO

Test: Are all Backends busy? YES Assign request to First free backend

NO

Randomly assign Request to a BE

Release host processor

Test: Is request a Modification? NO

YES

Compare request to list

Test: Request have Older Timestamp? YES Reject request

NO

Store in Modification Table:
1) Modifications Timestamp
2) Modifications Status

Seize backend

Process request to completion

Release backend

Test: Lull in processing At this backend? YES Send oldest Secondary Modification To each BE

NO

Loop until all User requests done

Terminate

Johnson & Thomas Model
FIGURE 3.4

Start

Seize host processor

Test: Is request a modification? YES Send request to all backends

NO

Test: Are all backends busy? NO Assign request to first free backend

YES

Randomly assign request to a backend

Release host processor

Process request to completion

Loop until all requests have been processed

Terminate

Ellis Model
enough times to send a copy to each of the backends in the network. At the backends, the request joins the jcb 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 requests 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

Start

Seize host processor

Test: Is request a modification?

YES

Assign request to Designated primary Backend

NO

Test: Are all Backends busy?

NO

Assign request to First free backend

YES

Randomly assign Request to a Backend

Release host processor

Seize backend

Process request to Completion

Release backend

Loop until all Requests are done

Send oldest modification To Secondary backend

Delay for buffering

Loop until all modification Requests have been sent

Terminate

Codasyl Model
FIGURE 3.6

Start

Seize host processor

Test: Is request a modification? YES Assign request to designated primary backend
NO

Test: Are all backends busy? NO Assign request to first free backend
YES

Randomly assign request to a backend

Release host processor

Seize backend

Process request to completion

Release backend

Test: Lull in processing at primary BE YES
NO Loop until all request done

Send oldest modification to secondary backends

Test: More pending modifications on list? YES
NO Terminate

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

Start

Seize host processor

Test: Is request a Modification
   YES Assign request to Backend designated Primary for that Record
   NO

Test: Are all Backends busy?
   YES Randomly Assign Request to a Backend
   NO Assign request to First free backend

Release host processor

Seize backend

Process request To Completion

Release backend

Test: Lull in processing At this backend? VRS
   NO Loop until all Requests are done

Test: Modification pending For this backend?
   YES Send oldest pending Modification to the Other backends
   NO

Terminate

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
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**

<table>
<thead>
<tr>
<th>Modification</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20% Modification</td>
<td>67127</td>
</tr>
<tr>
<td>35% Modification</td>
<td>68518</td>
</tr>
<tr>
<td>50% Modification</td>
<td>78739</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Backends</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>74055</td>
</tr>
<tr>
<td>4</td>
<td>70448</td>
</tr>
<tr>
<td>8</td>
<td>69880</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson &amp; Thomas</td>
<td>64794</td>
</tr>
<tr>
<td>Bernstein</td>
<td>66907</td>
</tr>
<tr>
<td>Hybrid</td>
<td>68225</td>
</tr>
<tr>
<td>Ellis</td>
<td>73261</td>
</tr>
<tr>
<td>Codasyl</td>
<td>84119</td>
</tr>
</tbody>
</table>

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

System Completion Time in Seconds

BERNSTEIN  JOHNSON & THOMAS  HYBRID  ELLIS  COPASYL
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
30% Modifications, 4 Backends

System Completion Time in Seconds

BERNSTEIN, JOHNSON, THOMAS, HYBRID, ELLIS, COPASTL
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
FIGURE 4.8

System Completion Time in Seconds

BERNSTEIN  JOHNSON &  HYBRID  ELLIS  CONASYL

3% Modifications, 8 Backends
50% Modifications, 8 Backends.

FIGURE 4.9
simulations is quite high, the validity of these results is suspect.

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

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 process the job stream faster with infinite channel speed. Although the trend was strong, it failed to achieve significance as measured by a T test.

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

1) Increasing the percent of modification commands will slow system processing.

2) Up to a point, increasing the number of backends in the network will speed processing of the job stream.

3) Three of the models, Bernstein, Hybrid, and Johnson and Thomas, process the job stream significantly faster than the other two models.
<table>
<thead>
<tr>
<th></th>
<th>CODASYL</th>
<th>ELLIS</th>
<th>JOHNSON &amp; THOMAS</th>
<th>HYBRID</th>
<th>BERNSTEIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>20% Mean</td>
<td>68574</td>
<td>61346</td>
<td>64597</td>
<td>6574</td>
<td>79778</td>
</tr>
<tr>
<td>20% St. Dev.</td>
<td>10795</td>
<td>13546</td>
<td>10795</td>
<td>1037</td>
<td>11593</td>
</tr>
<tr>
<td>35% Mean</td>
<td>58997</td>
<td>60785</td>
<td>58997</td>
<td>58991</td>
<td>77715</td>
</tr>
<tr>
<td>35% St. Dev.</td>
<td>2930</td>
<td>60785</td>
<td>2930</td>
<td>31039</td>
<td>11543</td>
</tr>
<tr>
<td>50% Mean</td>
<td>88520</td>
<td>74978</td>
<td>50547</td>
<td>72547</td>
<td>86066</td>
</tr>
<tr>
<td>50% St. Dev.</td>
<td>11045</td>
<td>4724</td>
<td>3864</td>
<td>12413</td>
<td>16022</td>
</tr>
</tbody>
</table>

*NOTE: Time is given in milliseconds.*
<table>
<thead>
<tr>
<th></th>
<th>BERNSTEIN</th>
<th>JOHNSON &amp; THOMAS</th>
<th>HYBRID</th>
<th>ELLIS</th>
<th>CODASYL</th>
</tr>
</thead>
<tbody>
<tr>
<td>20%</td>
<td>55784</td>
<td>59578</td>
<td>66908</td>
<td>68969</td>
<td>68245</td>
</tr>
<tr>
<td>MEAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST. DEV.</td>
<td>6538</td>
<td>4506</td>
<td>4596</td>
<td>10459</td>
<td>10025</td>
</tr>
<tr>
<td>35%</td>
<td>66977</td>
<td>56694</td>
<td>66779</td>
<td>69110</td>
<td>75141</td>
</tr>
<tr>
<td>MEAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST. DEV.</td>
<td>10023</td>
<td>4162</td>
<td>7943</td>
<td>8087</td>
<td>11525</td>
</tr>
<tr>
<td>50%</td>
<td>75912</td>
<td>75864</td>
<td>74143</td>
<td>82234</td>
<td>93641</td>
</tr>
<tr>
<td>MEAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST. DEV.</td>
<td>13169</td>
<td>11206</td>
<td>13795</td>
<td>7470</td>
<td>10125</td>
</tr>
</tbody>
</table>

* NOTE: Time is given in milliseconds
<table>
<thead>
<tr>
<th></th>
<th>Bernstein</th>
<th>Johnson &amp; Thomas</th>
<th>Hybrid</th>
<th>Ellis</th>
<th>Codasyl</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>20%</strong>&lt;br&gt;Mean</td>
<td>61482</td>
<td>63818</td>
<td>62362</td>
<td>69882</td>
<td>77068</td>
</tr>
<tr>
<td><strong>ST. DEV.</strong>&lt;br&gt;</td>
<td>15513</td>
<td>2946</td>
<td>14732</td>
<td>8320</td>
<td>8781</td>
</tr>
<tr>
<td><strong>35%</strong>&lt;br&gt;Mean</td>
<td>64338</td>
<td>63775</td>
<td>64166</td>
<td>64039</td>
<td>84298</td>
</tr>
<tr>
<td><strong>ST. DEV.</strong>&lt;br&gt;</td>
<td>7456</td>
<td>12802</td>
<td>10481</td>
<td>12603</td>
<td>10713</td>
</tr>
<tr>
<td><strong>50%</strong>&lt;br&gt;Mean</td>
<td>76001</td>
<td>53809</td>
<td>66509</td>
<td>78655</td>
<td>98029</td>
</tr>
<tr>
<td><strong>ST. DEV.</strong>&lt;br&gt;</td>
<td>14794</td>
<td>4535</td>
<td>4104</td>
<td>3526</td>
<td>10580</td>
</tr>
</tbody>
</table>

* NOTE: Time is given in milliseconds*
Figure 4.10

50% Modifications, 2 Backends

With Communication Delay

No Communication Delay
Figure 4.12

50K Modifications, 8 Backends
With Communication Delay
No Communication Delay
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 outperformed 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 its consistency anomaly. 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

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 Bernstein 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.
HYBRID I: All modifications sent to one primary backend.

HYBRID II: Modifications assigned 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 CoDasy 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 criticized for its complexity and the Hybrid model was criticized for having never been formally 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


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


**THE BERNSTEIN MODEL**

**ENTITY DEFINITIONS**

**PARAMETERS**

- PH1: TERMINAL ASSIGNMENT NUMBER
- PH2: NUMBER OF I/O REQUESTS NEEDED FOR EACH USER REQUEST
- PH3: SIZE OF MESSAGE BUFFER
- PH4: THE ABSTRACT CLOCK TIME
- PH5: MARKED 0 IF READ REQUEST
- PH6: MARKED 0 IF PRIMARY UPDATE REQUEST
- PH7: A COUNTER
- PH8: NUMBER OF REQUESTS MADE BY THE USER
- PH9: CHANNEL ID USED BETWEEN TWO BACKEND MACHINES
- PH10: TRANSMISSION SPEED CHANNEL IN BITS/SEC
- PH11: CHANNEL TRANSMISSION SPEED IN BITS/SEC
- PH12: TIME USED BY HINT OR BINT
- PH13: TIME USED BY THE MESSAGE SYSTEM
- PH14: 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->XH2: CURRENT TABLE POINTERS FOR BE1->BE8
- XM1->XM2: LAST TABLE POINTERS FOR BE1->BE8

**DEFINITIONS**

- UPDEI: PRIMARY UPDATE
- UPDE2: SECONDARY UPDATE
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
BECLN VARIABLE 10+PH5+PH15

TYPE FUNCTION RN2,D2
.80,0.1,100,1

20% UPDATE

TERM FUNCTION RN2,D8
.125,0.125/0.375,0.375/.5,0.5/.625,0.625/.75,0.75/.875,0.875/1.0,1.0/1.08

REOAD FUNCTION RN2,D10
.10,1/0.2,0.2/.3,0.3/.4,0.4/.5,0.5/.6,0.6/.7,0.7/.8,0.8/.9,0.9/1.0,1.0

REQO FUNCTION RN2,D10
.10,1/0.2,0.2/.3,0.3/.4,0.4/.5,0.5/.6,0.6/.7,0.7/.8,0.8/.9,0.9/1.0,1.0/1.0,1.0

LENGTH FUNCTION RN2,D2
.90,1,128/1.00,256

90% FIT IN ONE BUFFER

HOBE FUNCTION PH5,D8 HOST TO BE CHANNELS
1.91/2.22/3.33/4.44/5.55/6.66/7.77/8.88

BEDEV FUNCTION PH5,D8 BE TO DEVICE CHANNELS
1.11/2.22/3.33/4.44/5.55/6.66/7.77/8.88

USRQG FUNCTION RN2,D9 # OF REQUESTS BY A USER
.10,1/0.2,0.2/.3,0.3/.4,0.4/.5,0.5/.6,0.6/.7,0.7/.8,0.8/.9,0.9/1.1,1.1/1.3,1.3/

RNDM FUNCTION RN2,D4
.25,1/0.3,0.3/.5,0.5/.7,0.7/.9,0.9

INITIAL XH1-XH8,0 INITIALIZE SAVEVALUES 1-10 TO ZERO
INITIAL XF1-XF6,0 INITIALIZE XF1-8 TO ZERO
INITIAL XH11-XH14,0 INITIALIZE SAVEVALUES 11-14 TO ZERO

GENERATE 1500.,50.,15.,15PF,15PH GENERATE 15 TRANSX, AVE 1/1.5 SEC

PRIORITY 1

ASSIGN 1,PSTM,PM ASSIGN TERMINAL # IN PH1
ASSIGN 6,4,PM NUMBER OF BACKENDS IN SIMULATION
ASSIGN 6,2,PF PRIMARY/SECONDARY UPDATE CODE INIT 2 OR R
ASSIGN 8,FNSUSREQ,PH STORE # OF USER'S COMMANDS IN PH
ASSIGN 11,50000,PF TRANSMISSION SPEED TO CHANNEL
ASSIGN 12,50000,PF TRANSMISSION SPEED OF CHANNEL
ASSIGN 13,5,PH ADVANCE TIME FOR MINT OR SINT
ASSIGN 14,5,PF ADVANCE TIME FOR MESSAGE SYSTEM
QUEUE PH1 QUEUE FOR TERMINAL
SEIZE PH1 SEIZE THE TERMINAL
DEPART PH1 DEPART QUEUE FOR TERMINAL

MAIN LOOP FOR USER REQUESTS

MORE ADVANCE 1000,500 TIME TAKEN TO TYPE REQUEST
ASSIGN 8-,1,PH DECREMENT # OF REQUESTS MADE BY USER
QUEUE 100 QUEUE HOST/Terminal CHANNEL
SEIZE 100 TERMINAL-Host CHANNEL
DEPART 100 LEAVE QUEUE HOST-Terminal CHANNEL
ADVANCE VSTRMN,PM LINE TRANSMISSION
RELEASE 100 RELEASE CHANNEL
QUEUE 99 QUEUE FOR CPU
SEIZE 99 SEIZE HOST CPU
DEPART 99
ASSIGN 2,FNSREQ,PH ASSIGN # OF IC REQUESTS IN PH 2
ASSIGN 3,FNSLENGTH,PH LENGTH OF BUFFER TRANSMISSION
ASSIGN 4,FNSTYPE,PH ASSIGN UPDATE OR READ
ADVANCE PH13 MINT
ADVANCE PH14 MESSAGE SYSTEM
TEST E PH4,1,##2 IF UPDATE MARK PF6 A 1
ASSIGN 6,1,PF
ASSIGN 2,FNSRECORD,PF MAKE RECORD ASSIGNMENT
ASSIGN 3,C1,PF MAKE CLOCK TIME ASSIGNMENT
ASSIGN 5,1,PH

CHECK FOR A FREE MACHINE

LOCPA RELEASE 99 FREE HOST CPU
QUEUE FNSHOB CHUTE WAIT FOR HOST-BE CHANNEL
SEIZE FNSHOB CHUTE SEIZE HOST-BE CHANNEL
DEPART FNSHOB CHUTE
ADVANCE VSCHANL
ADVANCE 1 TIME TAKEN TO CHECK IF FREE
GATE SNE PH5,BEDB GO TO BEDB IF BE FREE
RELEASE FNSHOB RETURN CHANNEL
QUEUE 99
SEIZE 99 SEIZE HOST CPU
DEPART 99
ASSIGN 5-,1,PH
TEST OR PH5,PH6,FULL SEND TO FULL IF ALL BE'S ARE BUSY
TRANSFER LOCPA
FULL ASSIGN 5,FNSRNDM,PH RANDOMLY ASSIGN BE MACHINE
ADVANCE 1 ASSIGNMENT TIME
RELEASE 99 RELEASE THE HOST PROCESSOR

BACKEND HAS BEEN SELECTED

QUEUE FNSHOB
SEIZE FN$MOBE
DEPART FN$MOBE
ADVANCE VSCHANL
BEND RELEASE FN$MOBE
ADVANCE PM13
QUEUE PH5
ENTER PH5
DEPART PH5
BEGIN TEST E PM4+1,++,2
PRIORITY 3
TRANSFER ,REQ1
TRANSF PH7 IS A COUNTER

PROCESSING OF IC REQUESTS

LOOP8 TEST NE PM2,PH7,FINIO
TEST NE PM4+1,++,2
RELEASE PH5
RETURN BE CPU
QUEUE FN$BEDEV
SEIZE FN$BEDEV
DEPART FN$BEDEV
ADVANCE 92
RELEASE FN$BEDEV
TEST NE PM4+1,++,2
SEIZE PH5
ADVANCE 1
INCREMENT COUNTER

IO COMPLETED

FINIO TEST NE PF6,0,UPDT2
TEST E PF4+1,++,2
SAVEVALUE PM5,+,1,MT
RELEASE THE SECONDARY MODIFICATIONS

RETURN INFORMATION TO THE HOST AND TERMINALS

READ1 MSAVEVALUE PM5,PF9+,4,0,MT
ADVANCE PM13
RELEASE PH5
LEAVE PH5
QUEUE FN$MOBE
SEIZE FN$MOBE
DEPART FN$MOBE
ADVANCE VSCHANL
ADVANCE PM14
RELEASE FN$MOBE
QUEUE 99
QUEUE FOR HOST
PREEMPT 99,PR
PREEMPT HOST CPU
DEPART 99
ADVANCE PM13
HINT
RETURN 99
QUEUE 100
WAIT FOR CPU-TERMINAL CHANNEL
99 PREAMPT 100, PR
100 DEPART 100
101 ADVANCE VSTRMNL
102 RETURN 100
103 TEST E PH6, O, MORE
104 RELEASE PH1
105 EDIT TERMINATE 1
*
* ENTER REQUESTS IN MODIFICATION TABLE
*
*
106 REQ1 SAVEVALUE PH5 + 1, XH
107 ASSIGA 9, XH = PH5, PF
108 SAVEVALUE PH5, PF9, 4, PF, MX
109 TEST NE PH5 + 1, ONE
110 TEST NE PH5 + 2, TWO
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 SAVEVALUE 8, XH6 + 1, PH2, MX
118 SAVEVALUE 8, XH6 + 2, PF2, MX
119 SAVEVALUE 8, XH6 + 3, PF3, MX
120 TRANSFER , SKIP1
121 SEVEN SAVEVALUE 7, XH7 + 1, PH2, MX
122 SAVEVALUE 7, XH7 + 2, PF2, MX
123 SAVEVALUE 7, XH7 + 3, PF3, MX
124 TRANSFER , SKIP1
125 SIX SAVEVALUE 6, XH6 + 1, PH2, MX
126 SAVEVALUE 6, XH6 + 2, PF2, MX
127 SAVEVALUE 6, XH6 + 3, PF3, MX
128 TRANSFER , SKIP1
129 FIVE SAVEVALUE 5, XH5 + 1, PH2, MX
130 SAVEVALUE 5, XH5 + 2, PF2, MX
131 SAVEVALUE 5, XH5 + 3, PF3, MX
132 TRANSFER , SKIP1
133 FOUR SAVEVALUE 4, XH4 + 1, PH2, MX
134 SAVEVALUE 4, XH4 + 2, PF2, MX
135 SAVEVALUE 4, XH4 + 3, PF3, MX
136 TRANSFER , SKIP1
137 THREE SAVEVALUE 3, XH3 + 1, PH2, MX
138 SAVEVALUE 3, XH3 + 2, PF2, MX
139 SAVEVALUE 3, XH3 + 3, PF3, MX
140 TRANSFER , SKIP1
141 TWO SAVEVALUE 2, XH2 + 1, PH2, MX
142 SAVEVALUE 2, XH2 + 2, PF2, MX
143 SAVEVALUE 2, XH2 + 3, PF3, MX
144 TRANSFER , SKIP1
145 ONE SAVEVALUE 1, XH1 + 1, PH2, MX
146 SAVEVALUE 1, XH1 + 2, PF2, MX
147 SAVEVALUE 1, XH1 + 3, PF3, MX
148 TRANSFER , SKIP1
149 SKIP1 ADVANCE 1
150 TRANSFER , VOTE
*
* ROUTINE CALLED FOR MODIFICATION REQUESTS
* UPDT1 TEST E PF6.1, UPDT2 GO TO UPOT2 IF THIS IS NOT A PRIMARY UPOT
* SKIP5 ASSIGN 7, 1, PH A COUNTER
* SKIP6 TEST LE PH7, PH6, VOTIN IF COUNTER # OF BES, GO TO VOTIN
* TEST NE PH7, PH5, ELSE DETERMINE # OF BE TO BE UPOTED
* ASSIGN 15, PH7, PH TRANSFER 2+9
* ELSE ASSIGN 7+1, PH TRANSFER , SKIP6 INCREMENT COUNTER
* ASSIGN 7+1, PH SPLIT 1, 2+2
* TRANSFER , SKIP6

* QUEUE VSBECLN QUEUE FOR BE TO BE CHANNEL
* SEIZE VSBECLN SEIZE THE BE TO BE CHANNEL
* DEPART VSBECLN LEAVE THE BE TO BE CHANNEL QUEUE
* ADVANCE VSCHANL TIME TAKEN TO SEND MESSAGE ACROSS THE CHA
* RELEASE VSBECLN FREE THE BE TO BE CHANNEL
* ASSIGN 5, XP=PH15, PF ASSIGN INTO PF5 # OF LAST ENTRY IN TABLE

* SEARCH MODIFICATION TABLE

* LOOPM TEST NE PF5, 0, VOTEA IF THERE ARE NO ENTRIES VOTE ACCEPT
* TEST L MX=PH5 (PF9, 3), MX=PH15 (PF5, 3), VOTEA TIMESTAMP LESS-VOTE
* TEST E MX=PH5 (PF9, 2), MX=PH15 (PF5, 2), DECRE IF RECORD # DIFFERENT-
* THERE TEST E MX=PH15 (PF5, 41, 0), WAIT IF REQUEST PENDING - WAIT

* VOTEA ADVANCE 10 VOTING TIME
* QUEUE VSBECLN QUEUE FOR BE TO BE CHANNEL
* SEIZE VSBECLN SEIZE THE BE TO BE CHANNEL
* DEPART VSBECLN LEAVE THE BE TO BE CHANNEL QUEUE
* RELEASE VSBECLN FREE THE BE TO BE CHANNEL
* SAVEVALUE PH5+1, XF INCREMENT VOTE TALLYER
* TEST E XF=PH5, PH6 HOLD TILL ALL VOTES IN
* ADVANCE 10 TIME TAKEN TO WRITE OR MESSAGE
* SAVEVALUE PH5, 0, XF SET VOTE TALLYER BACK TO 0
* ADVANCE 1 DELAY NEEDED TO ASSEMBLE THE REQUESTS
* SAVEVALUE PH5+1, XF INCREMENT COUNTERT
* TEST E XF=PH5, PH6 WAIT UNTIL PRIMARY UPDATE IS COMPLETE
* QUEUE VSBECLN QUEUE FOR BE TO BE CHANNEL
* SEIZE VSBECLN SEIZE THE BE TO BE CHANNEL
* DEPART VSBECLN LEAVE THE BE TO BE CHANNEL QUEUE
* ADVANCE VSCHANL TIME TAKEN TO SEND MESSAGE ACROSS THE CHA
* RELEASE VSBECLN FREE THE BE TO BE CHANNEL
* ASSIGN 5, PH15, PH ASSIGN BE # INTO PH5
* ASSIGN 6, 0, PF MARK AS SECONDARY UPDATE
* QUEUE PH5 QUEUE FOR BE PARTITION
* ENTER PH5 ENTER BE PARTITION
* DEPART PH5
* TRANSFER , BEGIN GO TO BEGIN AND BEGIN SECONDARY UPDATINGS
* WAIT ADVANCE 5 TIME OUT 5 MSEC
* TRANSFER , THERE
* DECRE ASSIGN 5-1, PF DECREMENT TABLE POINTER
**200** VCTIN SAVEVALUE PH5+,1,XF
**201** TEST E XF=PH5,PH6
**202** TRANSFER +LABEL

**203** UPOTZ MSSAVEVALUE PH15,XH=PH15,4;0, NX
**204** LEAVE PH5
**205** RELEASE PH5
**206** ASSEMBLE VS MINUS
**207** TRANSFER ,ENDIT
START 800,,400
NORXREF ENC

**68** INCREMENT VOTE TALLYIER
**HOLD TILL ALL VOTES COUNTED**

MARK REQUEST AS COMPLETED
**LEAVE BE PARTITION**
**ASSEMBLE ALL SECONDARY UPDATES**

**** ASSEMBLY TIME = .04 MINUTES ****
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 REQUEST
PF2: THE SPECIFIC RECORD NUMBER
PH3: SIZE OF MESSAGE BUFFER
PH4: THE ABSOLUTE CLOCK TIME
PH5: MARKED O IF READ REQUEST
1 IF PRIMARY UPDATE REQUEST
2 IF SECONDARY UPDATE REQUEST
PF4: MARKED O IF UPDATE REQUEST HAS NOT BEEN SPOOLED
1 IF UPDATE REQUEST HAS BEEN SPOOLED
PH6: ID NUMBER OF THE BACKEND MACHINE BEING USED
PH7: NUMBER OF BACKEND MACHINES IN THE SIMULATION
PH8: A COUNTER
PH9: NUMBER OF REQUESTS MADE BY THE USER
PF10: TRANSMISSION SPEED TO CHANNEL IN BITS/SEC
PF11: CHANNEL TRANSMISSION SPEED IN BITS/SEC
PH12: TIME USED BY HINT OR BINT
PH13: TIME USED BY THE MESSAGE SYSTEM
PH14: 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
1214: 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

XM1->XM4: CURRENT TABLE POINTERS FOR BE1->BE4
XM11->XM14: 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
HALFWCRC 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
TRMNLFVARIABLE PH3=8/PF11=1000
CHANLFVARIABLE PH3=8/PF12=1000
LASTVARIABLE 10+PH5
INCREVARIABLE PH7+1
BECLNVARIABLE 10+PH5+PH15
TYPE FUNCTION RN2, 02
.80, 0/1.00, 1

20% UPDATE
TERM FUNCTION RN2, 08
RECD FUNCTION RN2, 010
.10, 1/.20, 2/.30, 3/.40, 4/.50, 5/.60, 6/.70, 7/.80, 8/.90, 9/1.0, 10
REQNO FUNCTION RN2, 010
.10, 1/.20, 2/.30, 3/.40, 4/.50, 5/.60, 6/.70, 7/.80, 8/.90, 9/1.0, 10
LENGTH FUNCTION RN2, 012
.90, 10/1.00, 256

90% FIT IN ONE BUFFER

MODE FUNCTION PH5, 08
1.91/2.92/3.93/4.94/5.95/6.96/7.97/8.98
BDEV FUNCTION PH5, 08
BE TO DEVICE CHANNELS
1.11/2.22/3.33/4.44/5.55/6.66/7.77/8.88
USRREQ FUNCTION RN2, 09
# OF REQUESTS BY A USER
.10, 1/.20, 2/.30, 3/.40, 4/.50, 5/.60, 6/.70, 7/.80, 8/.90, 9/1.0, 10

RNDM FUNCTION RN2, 04
.25, 1/.50, 2/.75, 3/.100, 4

INITIAL XH1-XH9, 0
INITIAL XH1-XH10, 0
INITIALIZE SAVEVALUES 1-10 TO ZERO
INITIALIZE SAVEVALUES 11-19 TO ZERO

GENERATE 1500, 50, 15s, 15PF, 15PM GENERATE 15 TRANSX, AVE 1/1.5 SEC
PRIORITY 1
ASSIGN 1, PH4 TERM, PH
ASSIGN TERMINAL # IN PH1
ASSIGN 6, 4, PH
NUMBER OF BACKENDS IN SIMULATION
ASSIGN 8, FNS, USREQ, PH
ASSIGN 11, 500000, PF
ASSIGN 12, 50000, PF
ASSIGN 13, 5, PH
ASSIGN 14, 5, PH
SAVEVALUE 9 + 1, XH
TEST E XH9, 15, +2
ASSIGN 9 + 1, PF
QUEUE PH1
SEIZE PH1
DEPART PH1

MAIN LOOP FOR USER REQUESTS

MORE ADVANCE 1000, 500
ASSIGN 9 + 1, PH
QUEUE 100
DEPART 100
ASSIGN 9 + 1, PH
QUEUE 100
DEPART 100

ADVANCE V$STRMNL
ADVANCE V$CHANL
RELEASE 100
QUEUE 99
SEIZE 99
DEPART 99

ASSIGN 2, FNSREGNO, PH
ASSIGN 3, FNSLENGTH, PM
ASSIGN 4, FNS_TYPE, PH
ADVANCE PM13
ADVANCE PH14
TEST E PH6, 1, #43
ASSIGN 2, FNSRECORD, PF
ASSIGN 3, CI, PF
ASSIGN 5, 1, PH

CHECK FOR A FREE MACHINE

LCPMA RELEASE 99
QUEUE FNSHOBE
SEIZE FNSHOBE
DEPART FNSHOBE
ADVANCE V$CHANL
ADVANCE 1
GATE SKE PH5, 8 EDB
RELEASE FNSHOBE
QUEUE 99
SEIZE 99
DEPART 99

ASSIGN 5 + 1, PH
TEST LE PH5, PH6, FULL
TRANSFER LCPMA
FULL ASSIGN 5, FNSRNDM, PH
ADVANCE 1
RELEASE 99

BACKEND HAS BEEN SELECTED
PROCESSING OF IO REQUESTS

035
036  LOOP B TEST NE PM2,PM7,FINIO IF IO DONE GO TO FINIO
037  TEST NE PH4+1,**2
038  RELEASE PH5
039  QUEUE FNSBEBDEV
040  SEIZE FNSBEBDEV SEIZE BE DEVICE FOR IO
041  DEPART FNSBEBDEV
042  ADVANCE 92 TIME FOR EACH IO
043  TEST E PH4+1,**4 IS REQUEST A PRIMARY UPDATE?
044  TEST NE PF4+1,**3 HAS THIS UPDATE ALREADY BEEN SPOOLED?
045  ADVANCE 30 IF NOT, SPOOL THE UPDATE
046  ASSIGN 4,1,PF MARK THE UPDATE AS SPOOLED
047  RELEASE FNSBEBDEV RELEASE DEVICE
048  TEST NE PH4+1,**2
049  SEIZE PH5 SEIZE THE BE CPU
050  ADVANCE 1 PROCESSING TIME
051  ASSIGN 7+1,PH INCREMENT COUNTER
052  TRANSFER ,LOOP B

IO COMPLETED

077
078  FINIO TEST NE PH4+1,UPOT1 IF UPDATE, GO TO UPOT1-CHANGE TABLE
079  TEST E PH4+2,READ IF QUERY ONLY COMMAND, GO TO READ

REQUEST IS A SECONDARY MODIFICATION

080
081  RECP H Release PH5 RELEASE THE BE PARTITION
082  LEAVE PH5 LEAVE SECONDARY UPDATE STORAGE
083  FINUP TEST G S*PH5,1,UPOT2 IF BE PARTITION EMPTY GO TO UPDATES
084  TRANSFER ,END T ELSE END TRANSACTION

RETURN INFORMATION TO THE HOST AND TERMINALS

099
100  READ ADVANCE PH13 BINT
101  RELEASE PH5
102  LEAVE PH5 LEAVE PARTITION IN BE
103  TEST E S*PH5,0,**2 IF PARTITION S EMPT Y, SPLIT
104  SPLIT 1,UPOT2 SPLIT AND SEND 1 TO UPOT2
105  QUEUE FNSHOB E WAIT FOR HOST-B E CHANNEL
SEIZE FN5H0BE
DEPART FN5H0BE
ADVANCE V5CHANL MESSAGE SYSTEM
ADVANCE PH1+4
RELEASE FN5H0BE QUEUE FOR HOST
QUEUE 99 PREEMPT FN5H0BE PREEMPT HOST CPU
PREEMPT 99, PR
DEPART 99 HINT
RETURN 99
QUEUE 100 WAIT FOR CPU-TERMINAL CHANNEL
PREEMPT 100, PR
DEPART 100
ADVANCE VSTRMN
RETURN 100
TEST E PH5,0,MORE MORE REQUESTS BY THIS USER?
RELEASE PH1 RELEASE OCCUPIED TERMINAL
TEST E PF91,1,>2 SKIP AROUND IF NOT LAST USER
GATE SE 9 NO ENTRY TILL ALL UPDATES HAVE BEEN COMPL
END IT TERMATADE 1

ENTER MODIFICATION REQUESTS IN BE MODIFICATION TABLE

UPDT SAVEVALUE PH5+1,XH INCREMENT # OF UPDATES SPOOLED
TEST NE PH5,1,ONE DETERMINE RS YOU ARE UPDATING ON
TEST NE PH5,2,TOO SO YOU CAN UPDATE APPROPRIATE TABLE.
TEST NE PH5,3,THREE
TEST NE PH5,4,FOUR
TEST NE PH5,5,FIVE
TEST NE PH5,6,SIX
TEST NE PH5,7,SEVEN
TEST NE PH5,8,EIGHT
EIGHT MSGSAVEVALUE 8,XH6,1,PH2,MX STORE THE NUMBER OF IO REQUESTS
MSGSAVEVALUE 8,XH6,2,PF2,MX STORE THE RECORD NUMBER
MSGSAVEVALUE 8,XH6,3,PF3,MX STORE THE ABSOLUTE CLOCK TIME OF UPDATE R
TRANSFER ,SKIP 1
SEVEN MSGSAVEVALUE 7,XH7,1,PH2,MX STORE THE NUMBER OF IO REQUESTS
MSGSAVEVALUE 7,XH7,2,PF2,MX STORE THE RECORD NUMBER
MSGSAVEVALUE 7,XH7,3,PF3,MX STORE THE ABSOLUTE CLOCK TIME OF UPDATE R
TRANSFER ,SKIP 1
SIX MSGSAVEVALUE 6,XH6,1,PH2,MX STORE THE NUMBER OF IO REQUESTS
MSGSAVEVALUE 6,XH6,2,PF2,MX STORE THE RECORD NUMBER
MSGSAVEVALUE 6,XH6,3,PF3,MX STORE THE ABSOLUTE CLOCK TIME OF UPDATE R
TRANSFER ,SKIP 1
FIVE MSGSAVEVALUE 5,XH5,1,PH2,MX STORE THE NUMBER OF IO REQUESTS
MSGSAVEVALUE 5,XH5,2,PF2,MX STORE THE RECORD NUMBER
MSGSAVEVALUE 5,XH5,3,PF3,MX STORE THE ABSOLUTE CLOCK TIME OF UPDATE R
TRANSFER ,SKIP 1
FOUR MSGSAVEVALUE 4,XH4,1,PH2,MX STORE THE NUMBER OF IO REQUESTS
MSGSAVEVALUE 4,XH4,2,PF2,MX STORE THE RECORD NUMBER
MSGSAVEVALUE 4,XH4,3,PF3,MX STORE THE ABSOLUTE CLOCK TIME OF UPDATE R
TRANSFER ,SKIP 1
THREE MSGSAVEVALUE 3,XH3,1,PH2,MX STORE THE NUMBER OF IO REQUESTS
MSGSAVEVALUE 3,XH3,2,PF2,MX STORE THE RECORD NUMBER
MSGSAVEVALUE 3,XH3,3,PF3,MX STORE THE ABSOLUTE CLOCK TIME OF UPDATE R
TRANSFER ,SKIP 1
TWO MSGSAVEVALUE 2,XH2,1,PH2,MX STORE THE NUMBER OF IO REQUESTS
MSAVEVALUE 2,XH2,2,PF2,MY STORE THE RECORD NUMBER
MSAVEVALUE 2,XH2,3,PF3,MY STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
TRANSFER, SKIP1
ONE MSAVEVALUE 1,XH1,1,PF2,MY STORE THE NUMBER OF IO REQUESTS
MSAVEVALUE 1,XH1,2,PF2,MY STORE THE RECORD NUMBER
MSAVEVALUE 1,XH1,3,PF3,MY STORE THE ABSOLUTE CLOCK TIME OF UPDATE RE
TRANSFER, SKIP1
ADVANCE 1 TIME TAKEN TO WRITE TO TABLE

ROUTINE CALLED TO PERFORM SECONDARY MODIFICATION REQUESTS

UPDT2 TEST L XH=VSLAST,XH=PH5,ENOIT CONTINUE IF UPDATES PENDING
SAVEVALUE VS=LAST+1,XH UPDATE TABLE POINTER LAST
ASSIGN 7,1,PH

LOOPC TEST L PM7,PM6,FINUP IF COUNTER OF BE GO TO FINUP
TEST L PM7,PM5,ELSE DETERMINE # OF BE TO BE UPDATED
ASSIGN 15,PM7,PH
TRANSFER, SKIP2
ELSE ASSIGN 15,Y=INCRE,PH
SKIP2 ASSIGN 9,VS=BECL,PH
ASSIGN 7+,1,PH INCREMENT COUNTER
SPLIT 1,LOOPC SEND OUT ANOTHER UPDATE TO NEXT BE

QUEUE PH5
ENTER PH5
DEPART PH5
SEIZE PH5 SEIZE THE BE CPU
ADVANCE PH13 BINT
RELEASE PH5
LEAVE PH5
ENTER 9 ENTER UPDATE PENDING STORAGE
QUEUE PH9 QUEUE FOR CHANNEL BETWEEN 2 BE'S
SEIZE PH9 BE TO BE CHANNEL
DEPART PH9
ADVANCE V=CHAN, MESSAGE SYSTEM
ADVANCE PH14
RELEASE PH9
QUEUE PH15
ENTER PH15
DEPART PH15
ASSIGN 1,X=VSLAST,PF
PRIORITY 3
SEIZE PH15 SEIZE THE BE CPU

= CHECK THE TIMESTAMP OF SAME RECORD MODIFICATIONS AT THAT BE
= TIMCK ASSIGN 5,XH=PH15,PF INITIALIZE CNTR TO FINAL ENTRY IN UPDT T1
= TEST NE PF5=0,CONTU IF THERE ARE NO ENTRIES, CONTINUE.
= LOOPD TEST L XH=PH5(PF1,3),XH=PH15(PF5,3),CONTU IS TIMESTAMP LESS?
= IF TIMESTAMP OLDER, DON'T PERFORM MODIFICATION

TEST E XH=PH5(PF1,2),XH=PH15(PF5,2),++5 IS IT THE SAME RECORD?
RELEASE PH15 RELEASE THE BACKEND PROCESSOR
LEAVE PH15 LEAVE THE BE PARTITION
LEAVE 9 LEAVE SECONDARY UPDATE STORAGE LIST
TRANSFER ;ENDIT
ASSIGN 5--; 1, PF
TEST & PF5, 0, LOOPD
CONTU ADVANCE PH13
ADVANCE 4++; 2, PH
CODE UPDATE AS SECONDARY COUNTER
ASSIGN 7--; 0, PH
ASSIGN 2, MX=PH5 (PF1, 1), PH
ASSIGN INTO PH2 THE 6 OF IO REQ
ASSIGN 5++; PH15, PH
ASSIGN PH15 INTO PH5
TRANSFER ,LOOP 8
START 800, 400
NOXREF
END

**** ASSEMBLY TIME = .04 MINUTES ****
**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 RWREQUEST
- **PF2**: MODIFICATION REQUEST NUMBER
- **PH3**: SIZE OF MESSAGE BUFFER
- **PF4**: 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 MINT 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, 42, 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**

- **XM1**: NUMBER OF MODIFICATION REQUESTS ACCEPTED BY HOST

**DEFINITIONS**

- **UPCAT1**: 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 51-58,2

TRMNAL PVARIABLE PH#8/PF11#1000
CHANL PVARIABLE PH#8/PF12#1000
LAST VARIABLE 10+PH5
INCRV VARIABLE PM7+1
BECNL VARIABLE 10+PH5+PH15
TYPE FUNCTION RNZ,0,2
  .80/0,1,0.0,1
  204 UPDATE
TERM FUNCTION RN2,0,8
REQND FUNCTION RN2,0,10
  .10,1/.20,2/.30,4/.40,5/.50,7/.60,9/.70,9.1/.80,12/.90,13/1.00,14
LNGTH FUNCTION RN2,0,2
  .90,128/.100,256
  908 FIT IN ONE BUFFER
MBOE FUNCTION PH5,0,8
  HOST TO BE CHANNELS
  1,91/2,92/3,93/4,94/5,95/6,96/7,97/8,98
BEDEV FUNCTION PH5,0,8
  BE TO DEVICE CHANNELS
  1,91/2,92/3,93/4,44/5,55/6,66/7,77/8,88
USRREG FUNCTION RN2,0,9
  # OF REQUESTS BY A USER
  .10,1/.20,2/.30,3/.40,4/.50,5/.60,6/.70,7/.80,13/.90,15/1.00,17
  
  RNOM FUNCTION RN2,0,4
  .25,1/.50,2/.75,3/1.00,4

INITIAL XM1-XM4,0
  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,FMSTERM,PH ASSIGN TERMINAL # IN PH1
   4 ASSIGN 6,4,PF NUMBER OF BACKENDS IN SIMULATION
   5 ASSIGN 8,FMISUSREG,PH STORE # OF USER'S COMMANDS IN PH8
   6 ASSIGN 11,50000,PF TRANSMISSION SPEED TO CHANNEL
   7 ASSNG 12,50000,PF TRANSMISSION SPEED TO CHANNEL
   8 ASSIGN 13,5,PM ADVANCE TIME FOR HINT OR BINT
   9 ASSIGN 14,5,PM 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 VSTMNL 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,FMISREQUDE,PH ASSIGN # OF ID REQUESTS IN PH2
  24 ASSIGN 3,FMISLENTH,PH LENGTH OF BUFFER TRANSMISSION
  25 ASSIGN 4,FMISYPTE,PH ASSIGN UPDATE OR READ
26 ADVANCE PH13 MINT
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 LOOP A RELEASE 99 FREE HOST CPU
31 QUEUE FN5H0BE WAIT FOR HOST-BE CHANNEL
32 SEIZE FN5H0BE SEIZE HOST-BE CHANNEL
33 DEPART FN5H0BE
34 ADVANCE V6SCHNL TIME TAKEN TO CHECK IF FREE
35 ADVANCE 1
36 GATE one PH5,BE86B GO TO BE86B IF BE FREE
37 RELEASE FN5H0BE RETURN CHANNEL
38 QUEUE 99
39 SEIZE 99 SEIZE HOST CPU
40 DEPART 99
41 ASSIGN 5,1,PH SEND TO FULL IF ALL BE'S ARE BUSY
42 TEST LE PH5,PH6,FULL RANDOMLY ASSIGN BE MACHINE
43 TRANSFER +,LOOP A
44 FULL ASSIGN 5,FNSR666,PH ASSIGNMENT TIME
45 ADVANCE 1 RELEASE THE HOST PROCESSOR
46 RELEASE 99

** BACKEND HAS BEEN SELECTED **

47 ENTER QUEUE FN5H0BE
48 SEIZE FN5H0BE
49 DEPART FN5H0BE
50 ADVANCE V6SCHNL
51 BE86B RELEASE FN5H0BE
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 LOOP B TEST NE PH2,PM7,READ IF IO DONE GC TO READ
59 TEST NE PH4,1,++2
60 RELEASE PH5
61 QUEUE FN38EDEV
62 SEIZE FN38EDEV
63 ADVANCE PH5 TIME FOR EACH IO
64 TEST E PH4,1,++4 IS REQUEST A PRIMARY UPDATE?
65 TEST NE PH4,1,++3 HAS THIS UPDATE ALREADY BEEN SPOOLED?
66 ADVANCE 30 IF NOT, SPOOL THE UPDATE
67 ASSIGN 4,1,PF MARK THE UPDATE AS SPOOLED
68 RELEASE FN38EDEV RELEASE DEVICE
69 TEST NE PH4,1,++2
70 ASSIGN PH5 SEIZE THE BE CPU
71 ADVANCE 1 PROCESSING TIME
72 ASSIGN 7,1,PH INCREMENT COUNTER
TRANSFER ,LOOPB

IO COMPLETED

RETURN INFORMATION TO THE HOST AND TERMINALS

READ ADVANCE PH13 BINT
RELEASE PH5
LEAVE PH5 LEAVE PARTITION IN BE
QUEUE PF5HOBEBE WAIT FOR HOST-BE CHANNEL
SEIZE PF5HOBEBE
DEPART PF5HOBEBE
ADVANCE V5CHNL
ADVANCE PH14 MESSAGE SYSTEM
RELEASE PF5HOBEBE
TEST E PH4,1,+++3
LOGICCK PF2 RESET THE LOGIC SWITCH TO FREE HELD TRANS
TRANSFER ,ENDIT
RETRAN QUEUE 99 QUEUE FOR HOST
PREEMPT 99,PR PREEMPT HOST CPU
DEPART 99
ADVANCE PH13 HINT
RETURN 99
QUEUE 100 WAIT FOR CPU-TERMINAL CHANNEL
PREEMPT 100,PR
DEPART 100
ADVANCE V5TRMN
RETURN 100
TEST E PH6,0,MORE MORE REQUESTS BY THIS USER?
RELEASE PH1 RELEASE OCCUPIED TERMINAL
ENDIT TERMINATE 1

ROUTE CALLED FOR MODIFICATION REQUESTS

UPDT ASSIGN 7,1,PH INCREMENT XH1—THE UPDATE COUNTER
SAVEVALUE 1+1,XH
ASSIGN 2,XH1,PF ASSIGN UPDATE # TO PF2
LOGICS PF2 SET LOGIC SWITCH PF2
LCOPC TEST LE PH7,PH6,WAIT IF COUNTER!=0 BE GO TO WAIT
ASSIGN 5,PH7,PH DETERMINE # OF BE TO BE UPDATED
INCREMENT COUNTER
ASSIGN 7+1,PH
SPLIT 1,ENTER SEND OUT TRANSX COPY TO BE UPDATED
TRANSFER ,LCOPC SEND THE PARENT TRANSX TO LCOPC

WAIT RELEASE 99 RELEASE THE HOST PROCESSOR
GATE LR PF2 WAIT UNTIL LOGIC SWITCH PF2 IS RESET
TRANSFER ,RETRAN GO TO RETRAN WHEN THE MODIFICATION REQUEST
HAS BEEN COMPLETED BY ONE OF THE BACKENDS

START 800,,100
ENC

**** ASSEMBLY TIME = .03 MINUTES ****
**COBOL MODEL**

**ENTITY DEFINITIONS**

**PARAMETERS**
- PH1: TERMINAL ASSIGNMENT NUMBER
- PF1: HALFWORD SAVEVALUE WHOSE NUMBER IS IO+PH5
- PH2: NUMBER OF I/O REQUESTS NECESSARY FOR EACH USER REQUEST
- 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, 32, 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**
- UPTAT1: PRIMARY UPDATE
- UPTAT2: SECONDARY UPDATE
- LAST: ROW OF MATRIX LAST USED IN UPDATING
- CURRENT: NUMBER OF UPDATES SPOOLED TO THAT BE
HALFWORL matrix

1: MODIFICATION TABLE FOR PRIMARY BACKEND

**M A T R I X  M N , 2 0 0 , 1**

STORAGES REPRESENT PARTITIONS IN EACH BACKEND

**S T O R A G E  S 1 - 5 8 , 2**
**T R M N L  F V A R I A B L E  P H 3 = 9 / P F 1 1 = 1 0 0 0**
**G A N L  F V A R I A B L E  P H 3 = 9 / P F 1 2 = 1 0 0 0**
**L A S T  V A R I A B L E  1 0 + P H 5**
**I N C R E  V A R I A B L E  P H 7 + 1**
**B E C N L  F V A R I A B L E  1 0 + P H 5 + P H 1 5**
**T Y P E  F U N C T I O N  R N 2 , 0 2**

**.8 0 , 0 / 1 . 0 0 , 1**

**2 0% UPDATES**

**T E R M  F U N C T I O N  R N 2 , 0 8**
**.1 2 5 , 1 0 1 / . 2 5 , 1 0 2 / . 3 7 5 , 1 0 3 / . 5 , 1 0 4 / . 6 2 5 , 1 0 5 / . 7 5 , 1 0 6 / . 8 7 5 , 1 0 7 / 1 . 0 , 1 0 8**

**R E Q N O  F U N C T I O N  R N 2 , 0 1 0**

**.1 0 , 1 / . 2 0 , 2 / . 3 0 , 4 / . 4 0 , 5 / . 5 0 , 7 / . 6 0 , 9 / . 7 0 , 1 1 / . 8 0 , 1 2 / . 9 0 , 1 3 / 1 . 0 0 , 1 4**

**L N G T H  F U N C T I O N  R N 2 , 0 2**

**.9 0 , 1 2 8 / 1 . 0 0 , 2 5 6**

**9 0% FIT IN ONE BUFFER**

**H O B E  F U N C T I O N  P H 5 , 0 8**

**H O S T  T O  B E  C H A N N E L S**

**1 , 9 1 / 2 . 9 2 / 3 . 9 3 / 4 . 9 4 / 5 . 9 5 / 6 . 9 6 / 7 . 9 7 / 8 . 9 8**

**B E D E V  F U N C T I O N  P H 5 , 0 8**

**B E T O  D E V I C E  C H A N N E L S**

**1 , 1 1 / 2 . 2 2 / 3 . 3 3 / 4 . 4 4 / 5 . 5 5 / 6 . 6 6 / 7 . 7 7 / 8 . 8 8**

**U S R E Q  F U N C T I O N  R N 2 , 0 9**

**# OF REQUESTS BY A USER**

**1 0 , 1 / . 2 0 , 2 / . 3 0 , 4 / . 4 0 , 5 / . 5 0 , 7 / . 6 0 , 9 / . 7 0 , 1 1 / . 8 0 , 1 2 / . 9 0 , 1 3 / 1 . 0 0 , 1 7**

**R K D R  F U N C T I O N  R N 2 , 0 4**

**.2 5 , 1 / . 5 0 , 2 / . 7 5 , 3 / 1 . 0 0 , 4**

**I N I T I A L  X H 1 - X H 9 , 0**

**I N I T I A L  X H 1 1 - X H 1 8 , 0**

INITIALIZE SAVEVALUES 1-9 TO ZERO

**I N I T I A L  X H 1 1 - X H 1 8 , 0**

**I N I T I A L I Z E  SAVEVALUES 1 1 - 1 8  TO  Z E R O**

**G E N E R A T E  1 5 0 0 , 5 0 , 1 5 , 1 5 P F , 1 5 P M**

**G E N E R A T E  1 5  T R A N S X , A V E  1 / 1 . 5 S E C**

**P R I O R I T Y  L**

**A S S I G N  1 , F N B T E R M , P H**

**A S S I G N  6 , 4 , P H**

**A S S I G N  8 , F N B T S U S R E Q , P H**

**A S S I G N  1 1 , 5 0 0 0 0 , P F**

**A S S I G N  1 2 , 5 0 0 0 0 , P F**

**A S S I G N  1 3 , 5 , P H**

**A S S I G N  1 4 , 5 , P H**

**A S S I G N  1 5 , 1 , X H**

**A S S I G N  1 4 , 1 , P F**

**D E P A R T  P H 1**

**S A V E V A L U E  5 4 , 1 , X H**

**T E S T  E  X H 5 , 1 5 , 4 , 2**

**T E S T  E  X H 5 , 1 5 , 4 , 2**

**I F  T H I S  I S  T H E  L A S T  T R A N S X , M A R K  I T**

**M A S K  A S  T H E  L A S T  T R A N S A C T I O N**

**M A I N  L O O P  F O R  U S E R  R E Q U E S T S**

**A S S I G N  9 , 1 , P H**

**A S S I G N  2 0 0 0 0 , 5 0 0**

**T I M E  T A K E N  T O  T Y P E  R E Q U E S T**

**A S S I G N  8 - 1 , P H**

**D E C R E M E N T  # O F  R E Q U E S T S  M A D E  B Y  U S E R**

**Q U E U E  1 0 0**

**Q U E U E  1 0 0**

**T E R M I N A L - H O S T  C H A N N E L**

**D E P A R T  1 0 0**

**L E A V E  Q U E U E  H O S T - T E R M I N A L  C H A N N E L**
ADVANCE VSTRMNL LINE TRANSMISSION
ADVANCE VSCHANL CHANNEL TRANSMISSION
RELEASE 100 RELEASE CHANNEL
QUEUE 99 QUEUE FOR CPU
SEIZE 99 SEIZE HOST CPU
DEPART 99
ASSIGN 2,FNSRECONC,PH ASSIGN # OF IO REQUESTS IN PH 2
ASSIGN 3,FNSLENGTH,PH LENGTH OF BUFFER TRANSMISSION
ASSIGN 4,FNSTYPE,PH ASSIGN UPDATE OR READ
ADVANCE PH13 HINT
ADVANCE PH14 MESSAGE SYSTEM
ASSIGN 5,1,PH
TEST NE PH4,1,AIGN IF UPDATE GO TO ASSIGN FOR ASSIGNMENT

CHECK FOR A FREE MACHINE

LOOP A RELEASE 99 FREE HOST CPU
QUEUE FN#HOBE WAIT FOR HOST-BE CHANNEL
SEIZE FN#HOBE SEIZE HOST-BE CHANNEL
DEPART FN#HOBE
ADVANCE VSCHANL
ADVANCE 1 TIME TAKEN TO CHECK IF FREE
GATE SNE PH5,BEDB GO TO BEDB IF BE FREE
RELEASE FN#HOBE RETURN CHANNEL
QUEUE 99
SEIZE 99 SEIZE HOST CPU
DEPART 99
ASSIGN 5,1,PH
TEST LE PH5,PH6,FULL SEND TO FULL IF ALL BE'S ARE BUSY
TRANSFER ,LOOFA
FULL ASSIGN 5,FNSRANDOM,PH RANDOMLY ASSIGN BE MACHINE
ASSIGN ADVANCE 1 ASSIGNMENT TIME
RELEASE 99 RELEASE THE HOST PROCESSOR

BACKEND HAS BEEN SELECTED

QUEUE FN#HOBE
SEIZE FN#HOBE
DEPART FN#HOBE
ADVANCE VSCHANL
BEBB RELEASE FN#HOBE
ADVANCE PH13 BINT
QUEUE PH5 QUEUE FOR BE PARTITION
ENTER PH5 ENTER BE PARTITION
DEPART PH5
TEST E PH4,1,##2
PRIORITY 3
SEIZE PH5
ASSIGN 7,0,PH PH7 IS A COUNTER

PROCESSING OF IO REQUESTS

LOOPB TEST NE PH2,PHT,FINIO IF IO DONE GO TO FINIO
TEST NE PH4,1,##2
RELEASE PH5
QUEUE FN#BEDEV
SEIZE FN#BEDEV SEIZE BE DEVICE FOR IO
69 DEPART  FN980EDEV  TIME FOR EACH IO
70 ADVANCE 52  IS REQUEST A PRIMARY UPDATE?
71 TEST E  PM4+1,0,04  HAS THIS UPDATE ALREADY BEEN SPOOLED?
72 TEST NE  PF4+1,0,03  IF NOT, SPOOL THE UPDATE
73 ADVANCE 30  MARK THE UPDATE AS SPOOLED
74 ASSIGN 4+10PF  RELEASE FN980EDEV  RELEASE DEVICE
75 RELEASE 7+1,0PF  TESTING TIME
76 TEST NE  PM4+1,0,02  INCREMENT COUNTER
77 SEIZE  PM9
78 ADVANCE 1
79 ASSIGN 7+1,0PH  INCREMENT COUNTER
80 TRANSFER 0,LOCPB

IO COMPLETED

ENTR MODIFICATION REQUESTS IN THE MODIFICATION TABLE

81 FINIO TEST NE  PHM+1,UPOT1  IF UPDATE, GO TO UPOT1-CHANGE TABLE
82 TEST E  PHM+2,READ  IF QUERY ONLY COMMAND, GO TO READ

REQUEST IS A SECONDARY UPDATE

83 SNOOP RELEASE  PM9
84 LEAVE  PM5  RELEASE THE BE PARTITION
85 FINUP TEST G  SM950,UPOT2  IF BE PARTITION EMPTY GO TO UPDATES
86 TRANSFER 0,ENDIT  ELSE END TRANSACTION

RETURN INFORMATION TO THE HOST AND TERMINALS

87 READ ADVANCE  PH13  BINT
88 RELEASE  PM9
89 LEAVE  PM5
90 QUEUE  FN930HBE
91 SEIZE  FN930HBE
92 DEPART  FN930HBE
93 ADVANCE  VSCHANL
94 ADVANCE  PH14  MESSAGE SYSTEM
95 RELEASE  FN930HBE
96 QUEUE  99
97 PREEMPT  99,PR  QUEUE FOR HOST
98 DEPART  99
99 ADVANCE  PH13  HINT
100 RETURN  99
101 QUEUE  100
102 PREEMPT  100,PR
103 DEPART  100
104 ADVANCE  VSCHANL
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  PH4+1,0,02  IF LAST TRANSACTION SPLIT
109 SPLIT  1,UPOT2
ENQIT TERMINATE 1

* AOC REQUEST TO MODIFICATION TABLE *

110 UPDT1 SAVEVALUE PH5+,1,XH INCREMENT # OF UPDATES SPOOLED
112 ONE MSAVEVALUE 1,XM1,1,XM2,MM STORE THE NUMBER OF TD REQUESTS
113 SKIP1 ADVANCE 1 TIME TAKEN TO WRITE TO TABLE
114 TRANSFER ,READ

* ROUTINE CALLED FOR SECONDARY MODIFICATION REQUESTS *

119 UPT2 SAVEVALUE 2+,1,XM
116 ASSIGN 14,0,PF
117 UPDT2 ASSIGN 5,1,PH
118 TEST L XM2,XM5,ENDIT CONTINUE IF UPDATES PENDING
119 ASSIGN 1,XM2,PF ASSIGN UPDATE TABLE ROW # INTO PF1
120 SAVEVALUE 2+,1,XH INCREMENT UPT LIST COUNTER
121 ASSIGN 7,1,PH

* SEND THE SECONDARY MODIFICATIONS *

122 LOOPC TEST LE PH7,PH6,UPDT2 IF COUNTER GT 0 OF BE GO TO FINUP
123 TEST NE PH7,PH5,ELSE GO TO ELSE IF CNTN = 0 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,BEGNL,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 V8CHANL
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,PH4,PH5(PF1,1),PH ASSIGN INTO PH2 THE # OF TD REQ
153 ASSIGN 5,PH15,PH ASSIGN PH15 INTO PH5
154 TRANSFER ,LOOPB
START 800,400
NOXREF
BLOCK

HYBRID MODEL

PARAMETERS

PH1: TERMINAL ASSIGNMENT NUMBER
PH2: NUMBER OF I/O REQUESTS NECESSARY FOR EACH USER REQUEST
PH3: SIZE OF MESSAGE BUFFER
PH4: MARKED 0 IF READ REQUEST
      1 IF PRIMARY UPDATE REQUEST
      2 IF SECONDARY UPDATE REQUEST
PH5: MARKED 0 IF UPDATE REQUEST HAS NOT BEEN SPOOLED
      1 IF UPDATE REQUEST HAS BEEN SPOOLED
PH6: ID NUMBER OF THE Backend MACHINE BEING USED
PH7: NUMBER OF Backend MACHINES USED IN THE SIMULATION
PH8: A COUNTER
PH9: NUMBER OF REQUESTS MADE BY THE USER
PH10: CHANNEL ID USED BETWEEN TWO Backend MACHINES
PH11: TRANSMISSION SPEED TO CHANNEL IN BITS/SEC
PH12: CHANNEL TRANSMISSION SPEED IN BITS/SEC
PH13: TIME USED BY MINT 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, 42, 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->XH9: CURRENT TABLE POINTERS FOR BE1->BE9
XH11->XH18: LAST TABLE POINTERS FOR BE1->BE9

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 VARIABLE PH3#8/PF11#1000
CHANL VARIABLE PH3#8/PF12#1000
LAST VARIABLE 10#PH5
INCRE VARIABLE PH7#1
BECLN VARIABLE 10PH5#PH15
TYPE FUNCTION RN2, 02
.80, 0/1.00, 1
20% UPDATE
TERM FUNCTION RN2, 08
.125, 101/25, 102/375, 103/5, 104/625, 105/75, 106/875, 107/1.00, 108
REQN FUNCTION RN2, 010
.10, 1/20, 2/30, 3/40, 4/50, 5/60, 6/70, 7/80, 8/90, 9/13.00, 14
LENGTH FUNCTION RN2, 02
.90, 128/1.00, 256
90% FIT IN ONE BUFFER
MODE FUNCTION PH5#08
HOST TO BE CHANNELS 1, 91/2, 92/3, 93/4, 94/5, 95/6, 96/7, 97/8, 98
BEDEV FUNCTION PH6#08
BE TO DEVICE CHANNELS 1, 11/2, 22/3, 33/4, 44/5, 55/6, 66/7, 77/8, 88
USRQ dispatch RN2, 09
# OF REQUESTS BY A USER
.10, 1/20, 2/30, 3/40, 4/50, 5/60, 6/70, 7/80, 8/90, 9/15.00, 17
RNDM FUNCTION RN2, 04
.25, 1/50, 2/75, 3/1.00, 4
INITIAL XM1=XM8, 0
INITIAL XM1=XM8, 0

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#PH, USRQ, PH
STORE # OF USER'S COMMANDS IN PH8
6 ASSIGN 11, 5000057, PH
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 FM1
QUEUE FOR TERMINAL
SEIZE \text{PH1} \quad \text{SEIZE THE TERMINAL}

DEPART \text{PH1} \quad \text{DEPART QUEUE FOR TERMINAL}

\text{MAIN LOOP FOR USER REQUESTS}

\begin{tabular}{ll}
13 & MORE \text{ADVANCE 1000, 500} \quad \text{TIME TAKEN TO TYPE REQUEST} \\
14 & ASSIGN \text{0-1, PH} \quad \text{DECREMENT \# OF REQUESTS MADE BY USER} \\
15 & QUEUE \text{100} \quad \text{QUEUE HOST/Terminal CHANNEL} \\
16 & SEIZE \text{100} \quad \text{TERMINAL-HOST CHANNEL} \\
17 & DEPART \text{100} \quad \text{LEAVE QUEUE HOST-Terminal CHANNEL} \\
18 & ADVANCE \text{VSTRMNL} \quad \text{LINE TRANSMISSION} \\
19 & ADVANCE \text{VSCANL} \quad \text{CHANNEL TRANSMISSION} \\
20 & RELEASE \text{100} \quad \text{RELEASE CHANNEL} \\
21 & QUEUE \text{99} \quad \text{QUEUE FOR CPU} \\
22 & SEIZE \text{99} \quad \text{SEIZE HOST CPU} \\
23 & DEPART \text{99} \\
24 & ASSIGN \text{2, FNSREQNO, PH} \quad \text{ASSIGN \# OF IO REQUESTS IN PH 2} \\
25 & ASSIGN \text{3, FNSLENGTH, PH} \quad \text{LENGTH OF BUFFER TRANSMISSION} \\
26 & ASSIGN \text{4, FNSTYPE, PH} \quad \text{ASSIGN UPDATE OR READ} \\
27 & ADVANCE \text{PH13} \quad \text{HINT} \\
28 & ADVANCE \text{PH14} \quad \text{MESSAGE SYSTEM} \\
29 & ASSIGN \text{5, 1, PH} \\
30 & TEST NE \text{PH4+1, FULL} \quad \text{IF UPDATE GO TO FULL FOR ASSIGNMENT} \\
\end{tabular}

\text{CHECK FOR A FREE MACHINE}

\begin{tabular}{ll}
31 & LOOPA RELEASE \text{99} \quad \text{FREE HOST CPU} \\
32 & QUEUE \text{FNSHOB} \quad \text{WAIT FOR HOST-6E CHANNEL} \\
33 & SEIZE \text{FNSHOB} \quad \text{SEIZE HOST-6E CHANNEL} \\
34 & DEPART \text{FNSHOB} \\
35 & ADVANCE \text{VSCANL} \\
36 & ADVANCE \text{1} \quad \text{TIME TAKEN TO CHECK IF FREE} \\
37 & GATE SNE \text{PH5, BDEB} \quad \text{GO TO BDEB IF BE FREE} \\
38 & RELEASE \text{FNSHOB} \quad \text{RETURN CHANNEL} \\
39 & QUEUE \text{99} \\
40 & SEIZE \text{99} \quad \text{SEIZE HOST CPU} \\
41 & DEPART \text{99} \\
42 & ASSIGN \text{5+1, PH} \\
43 & TEST LE \text{PH5, PH6, FULL} \quad \text{SEND TO FULL IF ALL BE'S ARE BUSY} \\
44 & TRANSFER \text{LOO} \\
45 & FULL ASSIGN \text{5, FNSRNHM, PH} \quad \text{RANDOMLY ASSIGN BE MACHINE} \\
46 & ASSIGN \text{ADVANCE 1} \quad \text{ASSIGNMENT TIME} \\
47 & RELEASE \text{99} \quad \text{RELEASE THE HOST PROCESSOR} \\
\end{tabular}

\text{BACKEND HAS BEEN SELECTED}

\begin{tabular}{ll}
48 & QUEUE \text{FNSHOB} \\
49 & SEIZE \text{FNSHOB} \\
50 & DEPART \text{FNSHOB} \\
51 & ADVANCE \text{VSCANL} \\
52 & BDEB RELEASE \text{FNSHOB} \\
53 & ADVANCE \text{PH13} \quad \text{BINT} \\
54 & QUEUE \text{PH5} \quad \text{QUEUE FOR BE PARTITION} \\
55 & ENTER \text{PH5} \quad \text{ENTER BE PARTITION} \\
56 & DEPART \text{PH5} \\
57 & SEIZE \text{PH5} \quad \text{SEIZE THE BE CPU} \\
\end{tabular}
ASSIGN 7+0,PH

PROCESSING OF IO REQUESTS

LOOP8 TEST NE PH2,PM7,FINIO IF ID DONE GO TO FINIO
TEST NE PH4,1,++,2
RELEASE PH5
QUEUE FN$BEDEV
SEIZE FN$BEDEV
DEPART FN$BEDEV
ADVANCE 52 TIME FOR EACH IO
TEST E PH4,1,++,3 HAS THIS UPDATE ALREADY BEEN SPOOLED?
ADVANCE 30 IF NOT, SPOOL THE UPDATE
ASSIGN 4+1,PF MARK THE UPDATE AS SPOOLED
RELEASE FN$BEDEV RELEASE DEVICE
TEST NE PH4,1,++,2
SEIZE PH5 SEIZE THE BE CPU
ADVANCE 1 PROCESSING TIME
ASSIGN 7+1,PH INCREMENT COUNTER
TRANSFER +LOOP8

IO COMPLETED

ENTER MODIFICATION REQUESTS IN MOD. TABLES

FINIO TEST NE PH4,1,UPDT1 IF UPDATE, GO TO UPDT1-CHANGE TABLE
TEST E PH4,2,READ IF QUERY ONLY COMMAND, GO TO READ

REQUEST IS A SECONDARY UPDATE

RECPU RELEASE PH5
LEAVE PH5
FINUP TEST G S=PH5,1,UPDT2 IF BE PARTITION EMPTY GO TO UPDATES
TRANSFER +ENDT ELSE END TRANSACTION

RETURN INFORMATION TO THE HOST AND TERMINALS

READ ADVANCE PH13 BINT
RELEASE PH5
LEAVE PH5 LEAVE PARTITION IN BE
TEST E S=PH5,0,++,2 IF PARTITIONS EMPTY, SPLIT
SPLIT 1,UPDT2 SPLIT AND SEND 1 TO UPDT2
QUEUE FN$SHDBE WAIT FOR HOST-BE CHANNEL
SEIZE FN$SHDBE
DEPART FN$SHDBE
ADVANCE VSCHNL MESSAGE SYSTEM
RELEASE FN$SHDBE
QUEUE 99 QUEUE FOR HOST
PREAMPT 99,PR PREEMPT HOST CPU
DEPART 99
ADVANCE PH13 HINT
RETURN 99
QUEUE 100 WAIT FOR CPU-TERMINAL CHANNEL
PREAMPT 100,PR
100  DEPART 100
101  ADVANCE VSYRMNL
102  RETURN 100
103  TEST E PHS, D, MORE
104  RELEASE PH1
105  ENDIT TERMINATE 1
106  * ADC REQUEST TO MODIFICATION TABLE
107  * UPDT1 SAVE VALUE PH5, 1, XH
108  * INCREMENT # OF UPDATES SPOOLED
109  TEST NE PH5, 1, ONE
110  TEST NE PH5, 2, THX
111  TEST NE PH5, 3, THREE
112  TEST NE PH5, 4, FIVE
113  TEST NE PH5, 5, SIX
114  TEST NE PH5, 7, SEVEN
115  TEST NE PH5, 8, EIGHT
116  EIGHT MS VALUE 8, XH, 1, PH2, MH
117  STORE THE NUMBER OF I/O REQUESTS
118  TRANSFER, SKIP1
119  SEVEN MS VALUE 7, XH, 1, PH2, MH
120  STORE THE NUMBER OF I/O REQUESTS
121  TRANSFER, SKIP1
122  SIX MS VALUE 6, XH, 1, PH2, MH
123  STORE THE NUMBER OF I/O REQUESTS
124  TRANSFER, SKIP1
125  FIVE MS VALUE 5, XH, 1, PH2, MH
126  STORE THE NUMBER OF I/O REQUESTS
127  TRANSFER, SKIP1
128  FOUR MS VALUE 4, XH, 1, PH2, MH
129  STORE THE NUMBER OF I/O REQUESTS
130  TRANSFER, SKIP1
131  THREE MS VALUE 3, XH, 1, PH2, MH
132  STORE THE NUMBER OF I/O REQUESTS
133  TRANSFER, SKIP1
134  TWO MS VALUE 2, XH, 1, PH2, MH
135  STORE THE NUMBER OF I/O REQUESTS
136  TRANSFER, READ
137  * ROUTINE CALLED FOR SECONDARY MODIFICATION REQUESTS
138  * UPDT2 TEST L XH = 0, LAST, XH = PH5, ENDIT CONTINUE IF UPDATES PENDING
139  * SAVE VALUE V, LAST, 1, XH
140  * UPDATE TABLE POINTER LAST
141  ASSIGN 7, 1, PH
142  * SEND THE SECONDARY MODIFICATIONS
143  LOOPC TEST LE PH7, PH6, FINUP
144  IF COUNTER GT # OF BE GO TO FINUP
145  TEST NE PH7, PH6, ELSE
146  ASSIGN 15, PH7, PH
147  TRANSFER, SKIP2
148  ELSE ASSIGN 7, 1, PH
149  TRANSFER, LOOPC
150  SKIP2 ASSIGN 9, V, BE, CNL, PH
151  ASSIGN 7, 1, PH
152  SPLIT 1, LOOPC
153  INCREMENT COUNTER
154  QUEUE PHS
155  ENTER PHS
DEPART PH5
SEIZE PH5
ADVANCE PH13
RELEASE PH5
LEAVE PH5
QUEUE PH9
SEIZE PH9
DEPART PH9
ADVANCE VSCHANL
ADVANCE PH14
RELEASE PH9
QUEUE PH15
ENTER PH15
DEPART PH15
SEIZE PH15
ADVANCE PH13
ASSIGN 4,2,PH
ASSIGN 7,0,PH
ASSIGN 1,0,VSLAST,PF
ASSIGN 2,0,PHS(PF1,1),PH
ASSIGN 5,PH13,PH
TRANSFER ,LOCPI
START 800..400
NOXREF ENC

**** 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 relative performance of five algorithms designed to keep redundant databases consistent. The throughput of each of the five algorithms is measured in an environment designed to saturate system resources.

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

Simulation models of the following algorithms are presented:

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 several environments and compared to each other. The advantages and disadvantages of each model are discussed in terms of performance, simplicity, and proven ability to maintain consistency.