TECHNICAL ARTICLE

SIMULATION 60:1,17-33

ISSN 0037-5497/92 $3.00 + .10
Printed in the United States of America

Simulation of Main Memory Database Recovery

Le Gruenwald
School of Computer Science
The University Of Oklahoma
Norman, Oklahoma 73019

Margaret H. Eich
Computer Science and
Engineering Department
Southern Methodist University
Dallas, TX 75275

In a main memory database (MMDB), the primary copy of the database may reside permanently in a volatile memory. When a system failure occurs, the database must be reloaded efficiently from archive memory into main memory. This paper presents four different reload schemes and the simulation models constructed to compare the algorithms. Simulation results indicate that the reload scheme based on frequency of data access gives the best overall performance in terms of transaction response time and system throughput.

Keywords: Database recovery, reload algorithms, reload schemes

Introduction

With the availability of large and inexpensive main memories, it becomes possible to place all or a major portion of the database in main memory instead of disks to improve the system performance. This leads to the designs of Main Memory Database Systems (MMDB) ([Ammann, 1985], [DeWitt, 1984], [Garcia-Molina, 1983], [Gruenwald, 1991], [Hagmann, 1986], [Kumar, 1991], [Lehman, 1987]).

The MMDB model used in this paper is based on *MARS(MAin memory Recoverable database with Stable log)*, a MMDB system designed at Southern Methodist University [Eich, 1987]. Its architecture is shown in Figure 1. The primary copy of the database is stored in main memory (MM) and its older version is stored in archive memory (AM) for backup purpose. Periodically, the database is checkpointed from AM to MM using the fuzzy checkpoint technique [Hagmann, 1986]. AM disks are organized using the *disk striping* technique [Salem, 1986] and consist of two areas: system data cylinders and user data cylinders. All updates take place in a stable memory (SM) which acts as a shadow memory. During normal transaction processing, dual address translation to MM and SM is used to detect if the value of the needed data is in SM. If so, then it takes precedence over that found in MM. At commit time, after-image records (AFIM) are copied from the shadow memory to MM and to the log buffer stored in SM. The SM also contains an archive memory directory and a checkpoinl bit map to indicate modified pages since the last checkpoint. When a page in the log buffer is full, it is copied onto the log disk. A database processor (DP) is

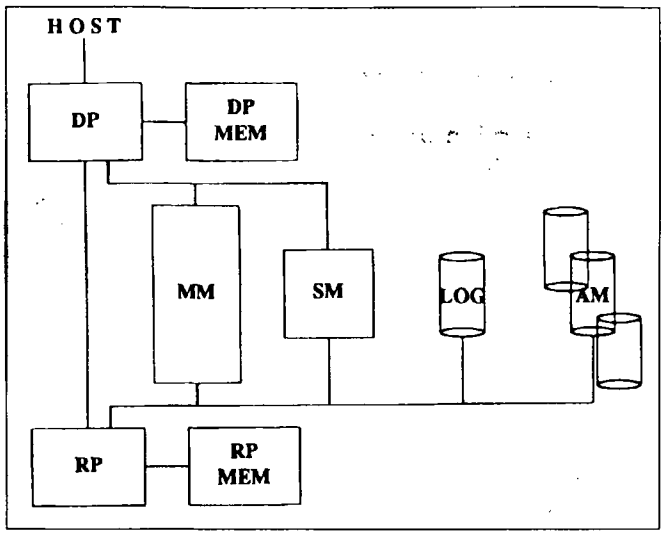


Figure 1. MARS Architecture

used to handle normal database processing, while a recovery processor (RP) is used to perform recovery processing acitivities. Each processor has its own memory (DP MEM/RP MEM) to facilitate its processing.

In a MMDB system, such as MARS, the primary copy of the database may be stored in a volatile main memory. The database needs to be reloaded from archive memory into main memory upon a system failure caused by power outage, preventative maintenance, software or hardware errors [Lehman, 1987]. Typically, power loss occurs 0.6 times per month [Controlled, 1987], and preventative maintenance once per month. If a system needs to process 1000 transactions per second and it is down for only 1/2 hour then 1,800,000 transactions may be lost. An efficient reload scheme must resume transaction processing quickly without degrading the overall system performance.

The purpose of this paper is to introduce several alternatives to perform the reload operation and to show how these alternatives can be simulated for comparison purpose. The rest of the paper is organized as follows. In Section 2 four different reload algorithms are presented. Section 3 describes the simulation models constructed to measure the performance of the algorithms. Section 4 highlights some results of the simulation experiments which indicate that the frequency-based reload algorithm outperforms other reload algorithms. Conclusions are given in Section 5.

Reload Algorithms

This section describes four different reload algorithms: *Ordered Reload, Ordered Reload with Prioritization, Smart Reload,* and *Frequency Reload.* These algorithms are different from one another based on the down time, reload prioritization, reload preemption, access frequency of data, and AM structure. Reload prioritization indicates the priority of data to be reloaded and reload preemption means that reload of some data can be suspended and replaced by reload of some other data.

Ordered Reload

This algorithm does not take reload prioritization, preemption, or access frequency into account. It reloads the data according to the order it is stored on AM. Its purpose is to reload the entire database in the shortest amount of time. This algorithm consists of the following steps.

- Step 1: (performed by DP): Reload the database into MM following the order in which the database is stored on AM on a cylinder basis. A cylinder basis means that cylinder 1 of all disks are reloaded in parallel. After cylinder 1 is memory resident then the reload of cylinder 2 starts and so on. Reload of cylinder 2 on a disk starts immediately after reload of cylinder 1 on the same disk is completed, even though reload of cylinder 1 on other disks may not be finished. This step is completed when the entire database is in MM.
- Step 2: (performed by RP in parallel with step 1): Copy to the shadow memory all AFIM records of committed transactions recorded on the log from the second to last begin checkpoint. These records are associated with a special recovery transaction.
- Step 3: Bring the system up.
- Step 4: (performed by RP in parallel with transaction processing): Copy the AFIMs records mentioned in Step 2 from the shadow memory to MM.

Note that due to the dual address translation to MM and SM during normal transaction processing mentioned earlier, the AFIMs from the log need not be applied to the checkpointed pages found in AM prior to bringing the system online.

Ordered Reload With Prioritization

This algorithm does not consider access frequency; but does consider reload prioritization and preemption. Its goal is to reload data that is needed immediately before other data so that the system can be brought up before the entire database is reloaded and waiting transaction response time can be reduced. The algorithm consists of the following steps:

- Step 1: Identify waiting transactions and their needed pages. From the information given by the AM directory, group these pages according to cylinders. Waiting transactions include not-yet-committed transactions and backlogged transactions when the system was down.
- Step 2: Reload system pages into MM on a cylinder basis. This means that cylinder I's of all disks are reloaded in parallel since cylinder I on each disk is assumed to store system pages.
- Step 3 (Performed by DP): Reload the rest of the database based on the following prioritization until the reload threshold is reached.

3.1. Priority 1 (highest): Reload pages needed by waiting transactions on a cylinder basis.

3.2. Priority 2: Reload the rest of the cylinders on all disks according to the order they are stored on disks.

• Step 4 (performed by RP in parallel with step 3): Copy to the shadow memory all AFIM records of committed transactions which are recorded on the log from the second to last begin checkpoint. All these AFIM records are associated with a special recovery transaction.

• Step 5: Bring the system up when the reload threshold is reached.

• Step 6: Reload the rest of the database based on the following prioritization until the entire database is in MM:

6.1. Priority 1 (highest priority): Reload pages needed by executing transactions on a demand basis. Executing transactions are those which arrive after the system resumes its execution.

6.2. Priority 2: Reload the rest of pages needed by waiting transactions on a cylinder basis.

6.3. Priority 3: Reload the rest of the cylinders on all disks according to the order they are stored on disks.

• Step 7 (performed in parallel to step 6): Copy all AFIM records mentioned in step 4 to MM.

When the reload threshold is reached, the system resumes its execution and priority 1 starts. At this point, reload preemption is in effect to ensure data of higher reload priority is brought into MM before data of lower reload priority by allowing the lower reload priority to be preempted by the higher one. Since cylinder is chosen to be the reload granularity for this algorithm, the reload preemption will not take place until the entire cylinder which is being reloaded is brought into MM. As a new transaction arrives, it demands some pages to be loaded into MM. This means that data of priority 1 has just been requested. The RP then checks to see if the current reload of a cylinder is completed. If yes, it switches the priority 2 to the background mode, stops executing priority 3, and starts reloading the data needed by the new transaction (priority 1) in the foreground mode.

Smart Reload

This algorithm uses prioritization, preemption, and access frequency. Its purpose is not only to reload data that is needed immediately before other data but also to reload data that is accessed more frequently before data that is accessed less frequently. Its motivation is to take advantage of *hot spots* which have been demonstrated to exist in many database applications ([Chou, 1985], [Copeland, 1986], [Gawlick, 1985], [Sacco, 1982], [Sacco, 1986]). *A hot set (or hot spot)* is a subset of the database that is frequently accessed. Reloading the hot sets first should reduce the number of page faults, and thus reduce response time of subsequent transactions. This algorithm does not attempt to predict what will be referenced next, instead it relies on the hot set concept to guarantee that the data reloaded is that with the highest probability of being referenced among data in the archive memory.

In this algorithm, at any point in time, reload of the most frequently accessed block into MM is expected, except for the case of reload on a demand basis. A block (page) is used as the reload granularity to reload in the precise order of frequency of access. This algorithm consists of the following steps:

• Step 1: identify waiting transactions and their needed pages the same way as with the *ordered reload* with prioritization algorithm.

• Step 2: Reload system pages into MM on a cylinder basis.

• Step 3 (performed by DP): Reload the rest of the database based on the following prioritization until the reload threshold is reached:

3.1. Priority 1 (highest priority): Reload pages needed by waiting transactions according to the decreasing order of access frequency of these pages.

3.2. Priority 2: Reload the rest of the database according to the access frequency. The highest frequency page is reloaded before the second highest frequency page and so on. At any time when reload of priority 2 takes place, the highest frequency page among the pages on each disk is searched and reloaded.

• Step 4 (performed by RP in parallel with step 3): Copy to the shadow memory the AFIM records of the committed transactions recorded on the log after the second to last begin checkpoint. Associate this with a special recovery transaction.

• Step 5: Bring the system up when the reload threshold is reached.

• Step 6: Reload the rest of the database based on the following prioritization until the entire database is memory resident.

6.1. Priority 1 (highest priority): Reload pages needed by executing transactions on a demand basis.

6.2. Priority 2:is the one assigned priority 1 in step 3.1.

6.3. Priority 3:is the one assigned priority 2 in step 3.2.

• Step 7 (performed in parallel to step 6): Copy all AFIM records in step 4 to MM.

Preemption of a lower priority reload by a higher priority reload is allowed; however there is no case in which an entire cylinder must be reloaded before preemption can take place. Instead, the preemption is in effect immediately right after the currently reloaded block is memory resident.

Frequency Reload

This algorithm takes reload prioritization, preemption, and access frequency into account. Its purpose is to reduce the total reload time as well as to improve throughput by trying to minimize the movement of disk

heads, reloading data that is needed immediately before other data, and taking advantages of *hot spots*. This algorithm works similarly to the *smart reload* algorithm except that it chooses cylinder instead of block to be its reload granularity and calls for a special AM structure, which is named *frequency AM structure*. Its intent is to approximate the smart reload but reduce the overhead and improve reload performance.

The frequency AM structure consists of two areas, system data cylinders and user data cylinders. The user data cylinders are arranged based on the decreasing order of page access frequency. Except for the case of demand reload, the algorithm ensures that, within each disk, lower numbered cylinders are reloaded before higher numbered cylinders. This means that more frequently accessed pages are brought into MM before less frequently accessed pages. Once data is placed on the disks in the correct order, the *frequency* reload algorithm works much the same as the *ordered reload with prioritization* algorithm.

Performance Analysis

There are many different approaches one can take to do performance analysis: analytic, benchmark, or simulation [Wilkinson, 1981]. As pointed out in [Deitel, 1984] and [Demurjian, 1987], it is usually difficult to find precise solutions for an analytical model if the model is complex and detailed. Therefore, this approach tends to simplify the problem in order to keep the formulae tractable [Wilkinson, 1981]. The benchmark approach studies the performance based on existing systems. The evaluator runs a benchmark, which is a production program, on the system that he or she wants to evaluate. As stated in [Deitel, 1984], "Benchmarks are not useful in predicting the effects of proposed changes, unless another system exists with the changes incorporated on which the benchmarks may be run". At present, MARS has not yet been built, thus the benchmark approach cannot be used. Since a detailed performance study is expected in this research, a detailed simulation model is chosen to be constructed for each reload algorithm.

Four simulation models are constructed to simulate the MARS system with a reload process incorporated. Each model has a different way to handle the reload process. The goal of the models is to measure the performance of the proposed reload algorithms when applied to MARS. This section describes the language chosen for simulation, resources and transaction representation, parameters used, performance measurements, and the overall simulation model for each algorithm.

Simulation Language

There are many programming languages on which a simulation model can be built. General purpose languages, such as Pascal, C, PL/I, can also be used. However, these languages were not invented for simulation; they require a lot of effort from simulators to implement features that are usually encountered in simulation: queuing, statistics gathering, parallelism, etc. To hasten the simulation process, a language that is designed for simulation such as SIMULA, SLAM II, should be used. Since SLAM II is available on SMU's IBM 3081, it is chosen to implement the simulation models for this research.

In this language, arrays are used to hold data in a simulation. Each entity created is associated with a one-dimensional array named ATRIB. Properties (attributes) of the entity are stored in this array. A one-dimensional array XX is available for retaining global data. SLAM II also provides a data structure called ARRAY. This structure consists of many one-dimensional arrays each of which is called an ARRAY row. These array rows are used to store global information for a simulation.

Resources

The simulation resources include LOG (log disk), DP (database processor), SM (stable memory), MM (main memory), AMI,...,AM5 (five disk units for archive memory), and RP (recovery processor). The RP is responsible for recovery activities, such as logging, checkpointing, reloading. This resource is assigned with four different allocation priorities. The highest priority is used when RP attempts to reload data due to page faults caused by the special recovery transaction mentioned in Section 2. The second highest priority is for reload due to page faults caused by normal transaction processing. The third highest is assigned to logging, committing, checkpointing, and reloading of data needed by waiting transactions. The lowest priority is used for reload of unneeded data in the background.

Transaction Representation

Each transaction is represented as an entity in SLAM II and has its own set of attributes. The values of these attributes might be different, but all transactions consist of the following attributes: *Transaction Identification* (a number starting from 1 to the total number of transactions examined in a particular run), *Multiprogramming Number* (a number from 1 to 20 since there are at most 20 transactions which can run concurrently), *Transaction Creation Time* (time at which the transaction enters the system), *Operation Number* (number of read or write operations a transaction performs. This number is generated using a uniform distribution between 5 and 10), *Operation* (read represented as 1.0 or write represented as 2.0; whether an operation is read or write depends on the read and write probabilities chosen), and *Page Number* (page that is accessed by the transaction operation; it is generated by using either an exponential distribution or uniform distribution which is selected before running the simulation). Note that the

Parameter	SLAM Variable	Default Value	Range
Number of AM disks	XX(1)	2	1..5
Number of Pages	XX(2)	1800	calculated XX(1)*XX(43)*XX(44)
Write Probability	XX(4)	20%	0%..100%
Read Probability	XX(5)	80%	100%-XX(4)
Multiprogramming Level	XX(7)	10	1..20
Total Transactions Examined	XX(9)	200	20..1000
Reload Threshold Percentage	XX(26)	60%	10%..100%
Number of Cylinders to Be Reloaded Per Disk	XX(43)	30	12..60
Blocks per Cylinder	XX(44)	30	15..60
Page Size	XX(45)	23476 bytes	11476..47476 bytes
Percentage of Transactions Commited Before System Goes Down	XX(52)	50%	10%..95%
Transfer Time	XX(59)	7.46 ms	3.65..15.09 ms
Mean of Exponential Distribution for Page Usage	XX(78)	1.5	0.5..2.5
Random Distribution Type for Page Usage	XX(79)	1	1,2 (1: Exponential, 2: Uniformed)

Table 1. Dynamic parameters.

Parameter	Meaning	SLAM Variable	Value
SM_ACCESS	Access an SM word	XX(21)	0.00011 ms
ALLOC_TM	Allocate a MM page	XX(22)	0.05 ms
AMREQ_TM	Request an I/O from AM	XX(23)	0.02 ms
PRETRAN	Preprocess a transaction	XX(24)	1.25 ms
PREOP	Preprocess an operation	XX(25)	0.005 ms
RELEASE_TM	Release an MM page	XX(27)	0.05 ms
BMAP_TM	Read until 1 in bit map	XX(28)	0.00011 ms
MM_ACCESS	Access an MM word	XX(29)	0.0001 ms

Table 2. Static parameters (part 1).

Parameter	Meaning	SLAM Variable	Value
SM_SEAR	AM address translation	XX(30)	0.5*XX(29)
MM_SEAR	MM address translation	XX(31)	3*XX(29)
MSEEK	Minimum seek time	XX(33)	3 ms
REC_SZ	SM or log record	XX(34)	12 bytes
ET_TM	End transaction	XX(35)	1.25 ms
INTIO_TM	Initiate log I/O	XX(36)	0.01 ms
LOGIO_TM	Write a log page	XX(37)	12 ms
COPY_TM	Copy an MM page to MM buffer	XX(38)	calculated XX(29)*2*XX(45)/XX(51)
AM_ACCESS	Access an MM page	XX(39)	calculated XX(57)+XX(58)+XX(59)
THRE_HOLD	Reload Threshold	XX(42)	calculated XX(26)*XX(1)*XX(43)*XX(44) (in smart algorithm) XX(26)*XX(1)*XX(43) (in other algorithms)
LOGPG_SZ	Log page size	XX(47)	2000 bytes
WORD_SZ	Bytes per word	XX(51)	4 bytes
TRACKS_CYL	Tracks per Cylinder	XX(56)	15
SEEK	Average seek time	XX(57)	16 ms
LATENCY	Average latency	XX(58)	8.3 ms
INDN_TM	Initial down time	XX(41)	5 ms
LOCK_TM	Get one lock	XX(54)	0.025 ms
UNLK_TM	Release one lock	XX(55)	0.025 ms
NUM_AFIMS	Number of committed AFIMS in log after the 2nd last checkpoint	XX(60)	10
HASH_TM	Hash AM directory for disk address	XX(65)	0.005 ms
AMDIR_TM	Get disk address from AM Directory	XX(68)	calculated XX(65)+(2*XX(21))
SIGNAL_TM	DP signals RP for a page fault	XX(69)	0.05 ms
CPU_POWER	Processor power	XX(75)	2 MIPS
FREQ_TM	Update Frequency Counter		calculated (3*1000/XX(75))+ (XX(21)*2)

Table 3. Static parameters (part 2)

last two attributes constitute a pair that must always go together. The first attribute in the pair indicates the operation to be performed on the page specified in the second attribute of the pair. The number of this pair is specified in the *Operation Number* attribute.

Simulation Parameters

The following tables present the dynamic and static parameters used in the simulation models. The parameter of random distribution type for page usage indicates the distribution function selected for generating page numbers used by transactions in a simulation run. This function is either exponential or uniform. If the former is chosen, its mean must also be provided. Most of the static parameters are adopted from existing literature. To simulate AM disks, the IBM 3380 disks are used [IBM, 1984]. The seek time, latency, transfer time, page size, number of blocks per disk, and number of tracks per cylinder are based on these disks. Time to allocate/release an MM page comes from [Salem, 1987]. Time to request an I/O and log page size are used in [Lehman, 1986]. Time that DP needs to signal RP for a page fault falls into the context switch range estimated in [Peterson, 1986]. Time to perform SM address translation is discussed in [Corti, 1991]. Time to perform

MM address translation is computed for the worst case in which 3 MM accesses are needed to access the segment table, page table, and to calculate the offset. The actual update of a frequency counter is assumed to take 3 instructions. The frequency counters are stored in the SM. To update a frequency counter, 2 accesses to SM must be done to get the previous value of the frequency counter and record its new value. The rest of the parameters are adopted from [Fan, 1988] with changes being made to reflect the difference in CPU power.

Performance Measurements

To study the performance of the proposed reload algorithms, the following measurements are obtained for each simulation run.

- Reload Time: this represents the total time an algorithm needs to reload the entire database into MM. In the ordered reload algorithm, since the entire database is reloaded before the system is brought up, the reload time only includes the I/O time. However, in the other algorithms, depending on the reload threshold, the system might resume its execution before the entire database is reloaded; the reload time therefore includes all elapsed time from the time the reload process starts until it finishes.
- System Unavailability: time during which the system is down. Given the system throughput when no failure occurs, this measurement dictates the number of transactions which must be backlogged when the system went down. It is undesired to have a long system unavailability. The difference between the reload time and the system unavailability is demonstrated in the following time line.
- Page Faults: this gives the number of page faults incurred by each algorithm.
- Transaction Response Time: this gives the mean of transaction response time. Together with this, a standard deviation is also obtained to see how the transaction response times differ from their mean value. Since the system may go down while a transaction is being processed, the response time of the transaction is evaluated from the time the transaction enters the system until it commits. In this case, the transaction response time also includes the down time.
- System Throughput: number of transactions committed per second. This is measured by dividing the total number of transactions committed by the time at which the last transaction commit is performed.

Among these measurements, the last two are the most crucial because they dictate the performance of the examined reload algorithm.

Model Descriptions

This section describes the simulation models used to measure the performance of the four proposed algorithms. Figure 3 shows a diagram of the simulation

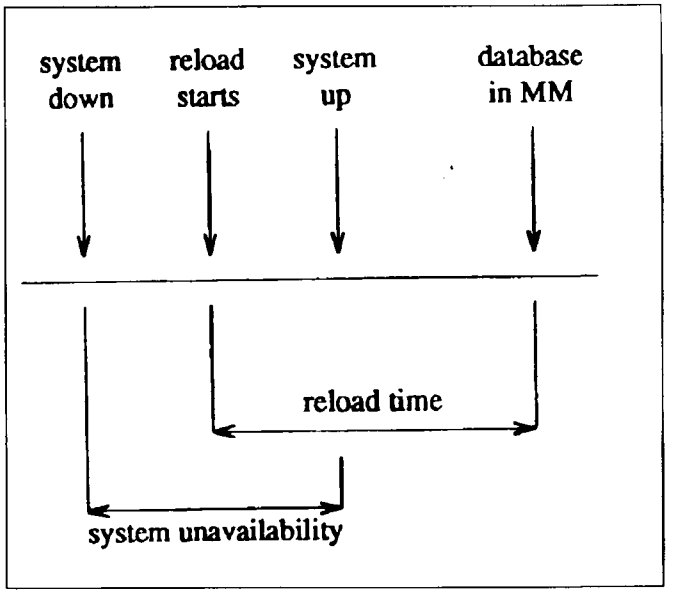


Figure 2. Time Line Diagram

model. Note that in this diagram, queues R1, R2, R3, and R4 stand for queues of different allocation priorities associated with the resource RP. All other resources, each has only one queue. Due to space limit, the diagram shows only one AM disk. The logical relationships among resources will not change if more AM disks are added.

Each model contains four components: initialization, transaction processing, checkpointing, and reloading. The reload component is only activated when system failure occurs. The details of these components are described in the following subsections. Due to the differences among the four proposed reload algorithms, there are also some modifications in the implementation of the components when applied to different reload algorithms.

Initialization Component

In this component, there are three major phases: 1) initialization of dynamic and static parameters and array rows, 2) frequency collection and priority queue construction, and 3) organization of the initial MM and AM structures to be used before a system failure occurs. Phase 1 is the same in all reload algorithms. Phase 2 is not needed in the ordered reload and ordered reload with prioritization algorithms. Phase 3 is the same in all algorithms except for the frequency algorithm.

Parameter and Array Row Initialization Phase

In this phase, all parameters are set to their initial values. The dynamic parameters are chosen by the evaluator for each simulation run while the static parameters are set to the values listed in Tables 2

through 3. Since SLAM II does not support dynamic data structures, arrays are essentially used to implement all data structures in this simulation. Each one-dimensional array is called one array row in SLAM II. In this phase, all array rows are initialized to 0.0's except for array row 38 which is set to 1.0's to indicate that all cylinders (or pages as in the smart algorithm) are not needed by waiting transactions and not yet reloaded at the beginning.

Frequency Collection and Priority Queue Construction Phase

In this phase, frequency of page access is gathered and priority queues of access frequencies are formed. The priority queues are needed to keep track which page has the highest frequency, second highest, and so on. The frequency reload algorithm uses this information to form its initial MM and AM structures in phase 3 while the smart reload algorithm uses this information in the reload component.

To simulate frequency collection, each simulation requires an extra run at the beginning called the first run. In this run, all XX(9) transactions that are intended to be examined in an actual run are generated in the same way mentioned in Section "Generation of Information for a Transaction." A frequency counter for each page is formed by counting the number of occurrences of the page. A user-defined array, FRE, is used to store these frequency counters. The frequency information is collected for all XX(9) transactions. In the actual simulation run (not the first one) the frequency counter array, FRE, is accessed to get the frequency information. In this simulation, the transactions that are executed are exactly the same as those that are generated to collect frequency. The knowledge of data reference behavior in transaction processing is therefore assumed to be accurate. However, later runs also evaluate the situation when this is not the case.

After the frequency collection is done, the frequency reload algorithm forms a priority queue to store pages in decreasing order of frequencies. The smart reload algorithm forms multiple priority queues, each of which is for one disk. The reason for this is that the smart algorithm always reloads a page that has the highest frequency of access on a disk. When reloading pages in parallel from multiple disks, a page of the highest access frequency on every disk is sought.

Initial AM and MM Structure Construction Phase

In this phase, the initial structures of the AM and MM are formed. These two memories are represented by array rows 31 and 32 respectively. Array row 31 is

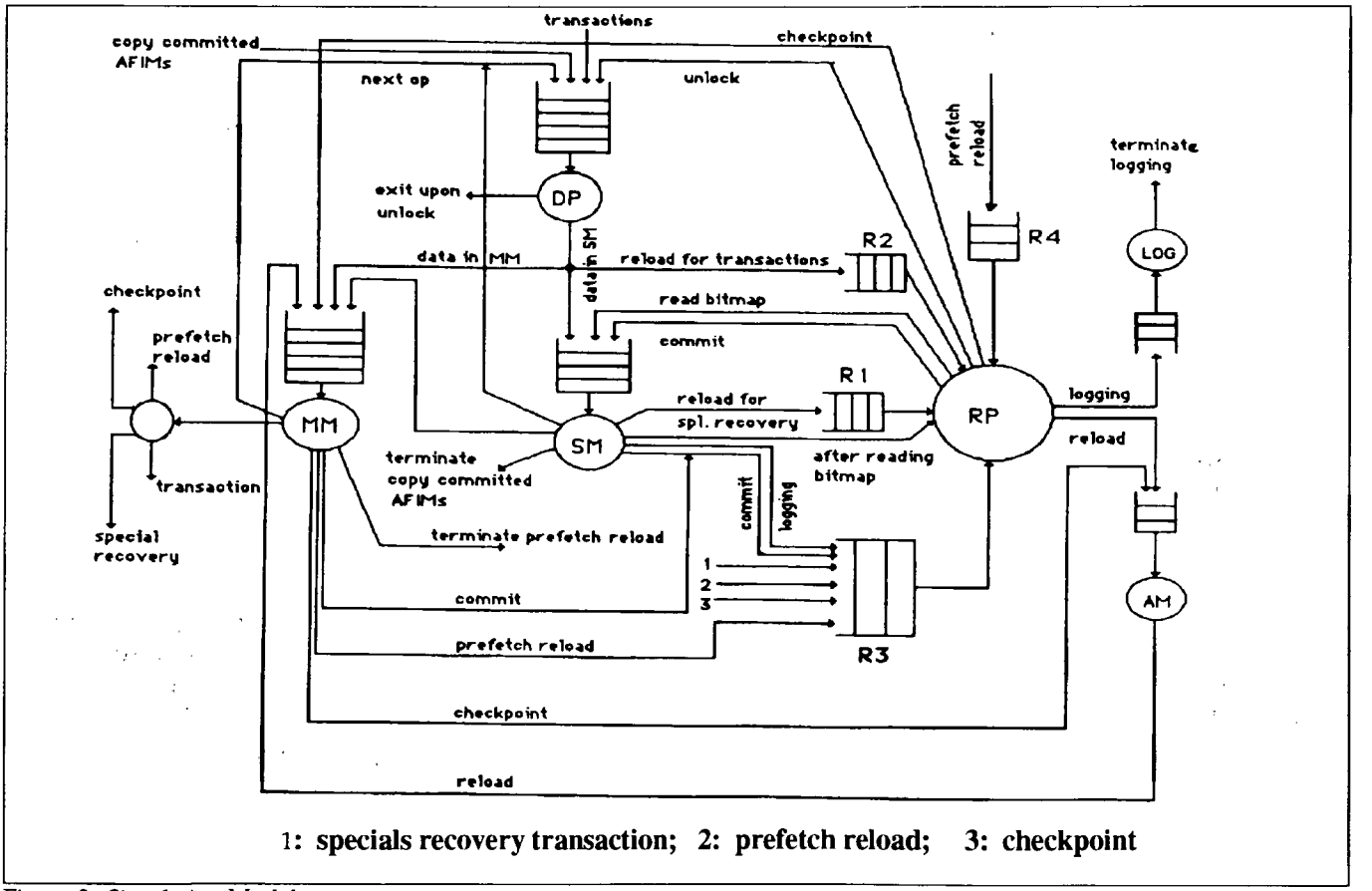


Figure 3. Simulation Model

equally divided into sections. Each section is used to store data for one AM disk. The initial structures of memories are used during the time before a system failure occurs. When a system failure takes place, the initial AM remains the same but the MM structure changes as the reload process comes into existence. The initial structures of MM and AM are the same when applied to the ordered reload, ordered reload with prioritization, and the smart reload algorithms. They do not follow any particular memory structure. Each location of the arrays that are used to represent MM and AM is filled with a page number in such a way that page number *i* is stored in location i. These structures are formed differently when applied to the frequency reload algorithm. Their structures are organized following the *frequency AM structure* model as described in Section 2 with the information in the priority queue of access frequencies formed in phase 2.

Transaction Processing Component

The component is created using a CREATE statement in SLAM II. Each transaction is represented by an entity. The number of transactions that can be executed concurrently in the system (multiprogramming level) is specified by the user. The CREATE statement is then used to generate those many entities at the same time starting at time unit 0. Each transaction then goes through several phases until it commits. Following are the descriptions of these phases.

Generation of Information for a Transaction

In this phase, each transaction is assigned its creation time which is the time it enters the system, transaction identification number, multiprogramming number, number of operations it is going to perform, operations, and pages on which operations are performed. The creation time is obtained from the SLAM global variable TNOW which gives the system current time. This attribute is collected later to obtain transaction response time. The transaction identification number is 1 greater than the transaction identification number assigned thus far. The first number assigned has a value of 1. The multiprogramming number is in the range of [1,XX(7)] where XX(7) is the multiprogramming level chosen for a particular simulation run. The number of operations a transaction is going to perform is generated by using a uniform distribution function for values between 5 to 10. This transaction size assumption is similar to the one used in [Wilkinson; , 1981]. The operations (reads or writes) are generated using a feature called probabilistic branching in SLAM II in which the probabilities of reads and writes are specified through two global variables XX(4) and XX(5). Note that these probabilities are dynamic parameters; thus they can be varied from run to run as desired.

Each operation generated is associated with a page number. These pages are obtained by using either an exponential distribution function or a uniform distribution function. If the exponential distribution function f(X) for page usage is chosen a mean must also be supplied. The value of X at which the cumulative distribution is 99% is called the cut-off point. The range between 0 and the cut-off point is divided into intervals. The number of the intervals is equal to the number of pages examined in a simulation run. This means that if an X falls into interval *i* then it is associated with page number *i*. If a uniform distribution function is chosen for page usage, then page numbers are formed uniformly between 1 and the maximum number of pages examined in a simulation run. There is no time associated with this phase.

Preprocessing and Locking Phase

In this phase, the database processor (DP) preprocesses the transaction and tries to obtain all needed locks for the transaction. Note that in our system, two-phase locking concurrency control is implemented with page level locking and preclaiming of all resources. If all locks cannot be granted, the transaction must wait in a gate whose identification number is the same as the transaction's multiprogramming number. A gate is a special feature in SLAM II. It allows entities to wait until it is opened by some other entities. The transaction that must wait in a gate is said to be blocked. It can only proceed if the gate is opened by some other transaction. Once the gate is opened, the transaction must go through the same process again to try to obtain all needed locks. This process of obtaining locks is repeated until the transaction is provided with all its locks.

Operation Processing Phase

Once the transaction passes the previous phase, each of its operations must be processed (no abnormal aborts are assumed to exist). The DP preprocesses an operation and performs SM and MM address translations in parallel for the page on which the operation is performed. The address of a page in this simulation is, in fact, the number of the page. MM and SM are implemented using two array rows 32 and 33 in SLAM II. The address transactions are done by searching the array rows for the page number. The SM address translation has a higher precedence than the MM address translation. If the page is found in SM, then if the associated operation is a read, the DP will read one SM word (AFIM). If the operation is a write, the DP will write an entire SM record onto SM, and set the corresponding bit in the bit map to 1 to indicate that the page has been modified. If the page is not found in SM but found in MM, then if the operation is a read, the DP will read one MM word. If the operation is a write, then copy the page number to the SM and continue processing via SM. If the page is found in neither SM nor MM, a page fault

occurs. How to bring this page into MM is a part of the reload component which will be described later. The operation processing is delayed until the needed page is reloaded. The same process is repeated for every operation until all operations performed by the transactions are processed. Note that when applied to the frequency and smart algorithms, this phase also incurs a frequency calculation overhead. In these two algorithms, after the DP preprocesses in operation, it must also update the frequency counter associated with the page on which the operation is will performed.

Transaction Commit Phase

There are two processes running in parallel in this phase: transaction committing and logging. These two activities are done by the recovery processor (RP). For transaction commit, if the transaction is read-only, the next phase is taken. Otherwise, the RP writes a BT record onto the log buffer to indicate the begin of the transaction, copies the transaction's shadow records from SM to the log buffer, and writes a record ET onto the log buffer to indicate the end of the transaction. The RP then updates the bit map and copies the transaction's AFIMS from SM to MM. The locations in the SM that are occupied by the transactions are then freed. For logging, if a log buffer page is full, the RP requests an I/O to flush that page to the log disk. The RP repeats the same process for all full log pages. The logging entity is then terminated.

Unlocking Phase

After the transaction commits, all its locks are released one by one by the DP. A FORTRAN event is called at this point to actually perform this process. After this is done, the transaction response time is collected on this transaction. The number of transactions committed is also accumulated.

After Unlocking Phase

Once a transaction finishes all phases listed above, it is said to completely finish its processing. The entity that is used to represent the transaction can now be used to represent another transaction. However, at this point a decision must be made. If the number of transactions committed is equal to the number of transactions committed before the system went down, the system is brought down to simulate a system failure. The reload component then takes place at this point. Otherwise, if the number of transactions generated is smaller than the number of transactions that must be examined in this simulation run, the entity is routed back to the first phase to generate information for next transactions. The same process is then repeated. If the number of transactions committed is equal to the number of transactions that must be examined then the system throughput is collected at this point (note that the system keeps going until the reload is completed). The system throughput is evaluated by dividing the number of transactions committed by the current system time TNOW. Since the maximum number of transactions to be created in a simulation run equals the number of transactions to be examined in the run, there will not be other transactions executing when the system throughput is collected.

When the transaction processing is activated after a system failure occurs, the special recovery transaction must be given immediate attention followed by the waiting transactions, before new transactions can be created. Whether all three types of transactions are allowed to be in the system at the same time depends on the multiprogramming level and the number of waiting transactions. When the system resumes its execution after a system failure, for each entity passed through this component, the following checks are done. If the entity represents the special recovery transaction, the commit for this transaction then takes place. If the waiting file described in the reload component is not empty, then a waiting transaction is removed from this file and its attributes are copied to the current entity. This means that the current entity now represents a waiting transaction. The waiting transaction skips the first phase of the transaction processing component described in the Section"Generation of Information for a Transaction." and proceeds from the second phase on. This process is repeated for all waiting transactions in the waiting file until the waiting file is empty. At this point, if the number of transactions created so far does not exceed the number of transactions to be examined in a simulation run XX(9), then new transactions are created. Each new transaction goes through all phases described in this component.

Checkpoint Component

In MARS, fuzzy checkpoint is used to checkpoint modified pages from MM to AM. A bit map stored in SM keeps track of modified pages. The checkpoint component is created through a CREATE statement starting at time until 10. This process starts by recording a record BC in the log buffer to indicate the beginning of the checkpoint The bit map is then searched for a 1 which indicates that the corresponding page has been modified since the last checkpoint. If a 1 is found, the RP copies the page from MM to MM buffer, and requests a write to the appropriate AM disk. The page is then written onto the AM disk and the page buffer is released. The checkpoint process is then repeated until all locations in the bit map are checked. The RP then writes a record CE to indicate the end of the checkpoint. One checkpoint process is said to have completed. There is a delay of 30 units of time until the next checkpoinl process begins. To simulate the next checkpoint process, the entity is routed back to the beginning of the checkpoinl component.

Reload Component

In this component, there are three different phases: initial work upon a system failure, reloading due to prefetch, and reloading due to page faults. The second phase takes place when a system failure occurs and is also going on in the background after the system is brought up as in the case of frequency, ordered reload with prioritization, and smart algorithms. The third one only comes into existence when the system resumes execution. These three phases are described in the following paragraphs.

Initial Work Upon a System Failure Phase

At the time when a system failure occurs, all resources must be made inactive, and all active transactions must be kept somewhere so that they can be restarted later. To simulate this situation, PREEMPT statements are used. These statements preempt the resources that are currently in use. Each preempted entity is checked to see if it is a waiting transaction or a checkpoint process. Waiting transactions are those that were active when the system went down. The identification is made by examining the multiprogramming number of the entity. The multiprogramming numbers assigned to transactions have the values in the range of [1, XX(7)] where XX(7) is the multiprogramming level chosen for a particular simulation run. The multiprogramming number of a checkpoint entity is 0. Note that there is only one checkpoint process in the system at any time.

If a preempted entity is a waiting transaction, it together with all of its attributes are stored in a file called waiting file. If it is a checkpoint process, its attributes will not be kept. The preempted entity is then terminated from the simulation. After all resources are preempted, all queues associated with resources are then emptied. The main memory is also emptied to simulate its volatility. The lock matrix is reinitialized and all multiprogramming gates that are used to block transactions in simulating concurrency control are reopened.

In all algorithms, except for the ordered reload algorithm, data needed by waiting transactions must be identified. A FORTRAN event is called to perform this. Array row 38 is used to keep track of which cylinders (or pages in the smart algorithm) are needed by waiting transactions and which cylinders are already reloaded. Array location (38,i) has a value of 2.0 if the cylinder i (or page i in the smart algorithm) is already reloaded; a value of 0.0 if the cylinder is not needed by waiting transactions and not yet reloaded. At this time, since reloading is not yet started, and all locations of array row 38 have been initialized to 1.0's by the initialization component discussed in Section 3.6.1.1., the only task to do is to set array location (28.i) to 0.0 for every cylinder i (or page i in the smart algorithm) needed by waiting transactions. In all reload algorthims, the system is then delayed for an INDN TM units of time before the next phase, reload due to prefetch, can be entered. Following is an example of the array 38 formed at this point for a simulation which has totally 6 cylinders to be reloaded on all disks. In this example, cylinders 2 and 4 are needed by waiting transactions and are not reloaded yet; cylinders 1,3,5 and 6 are not needed by waiting transactions and not reloaded yet.

1	2	3	4	5	6
1.0	0.0	1.0	0.0	1.0	1.0

Figure 3. An example of array 38.

Reload Due to Prefetch Phase

In this phase, XX(1) concurrent entities are generated to simulate XX(1) reload processes for XX(1) striped disks. Each reload process reloads data residing on a disk. Each disk is associated with a disk number starting from 1. To reload a cylinder, the RP initializes an I/O request to the I/O channel associated with a disk. The I/O channel is then responsible for reading the cylinder from the associated AM disk and writing the cylinder onto the main memory. Once a cylinder is memory resident, the reload process then starts reloading another cylinder. The physical main memory address for each page reloaded is formed by increasing the current main memory physical address used thus far by 1. Therefore cylinders on the same disk might not reside in consecutive locations in the main memory.

There are some differences in implementing this phase for different reload algorithms. The specific implementation of this phase for each algorithm is described below.

• Ordered Reload Algorithms: when this phase is first activated, cylinder 1 which is assumed to carry system data on each disk is reloaded into the main memory. The seek time used in reloading cylinder 1 on each disk is the average seek. After all cylinder 1's are memory-resident, one more concurrent process is generated to copy the AFIMS from the 2nd from the last checkpoint in the log to the shadow memory. The number of these AFIMS is assigned to the variable NUM_AFIMS. The default value is 10. The page numbers for these AFIMs are generated using either exponential distribution or uniform distribution for page usage as described in the transaction processing component. The copy is done by the DP in parallel to the reload processes. The AFIMs are associated with a special recovery transaction whose identification number is a negative number -1000. Each reload process, at the same time, continues reloading cylinder 2, cylinder 3, and so on for a disk. The minimum seek time is used in reloading these adjacent cylinders. Note that when one disk reloads a cylinder, say 3, another disk might still reload cylinder 2. When all cylinders are reloaded, that is the entire database is

memory-resident, all entities are terminated except one and the system is brought up. The surviving entity is used to simulate the processing part that takes place after the system resumes its execution. Note that in SLAM II, the simulation is terminated if either there is no entity alive in the system or the simulation time is up. How the system is brought up and what is going to take place after the system is brought up must be simulated; the simulation cannot be terminated here. Therefore one entity must be kept active at this time. The reload time is measured for the simulated duration from the start to the end of this component.

When the system is brought up from a system failure, the only surviving entity is split into a number of concurrent entities: one entity to represent the special recovery transaction, XX(7)-1 entities to represent normal transaction processing (recall XX(7) is the multiprogramming level), and one entity for checkpointing. The checkpoint entity will be initialed after a delay of 10 units of time. The transaction processing entities which include the special recovery transaction and normal transactions are routed to the transaction processing component. The checkpoint process is branched to the beginning of the checkpoint component.

• Ordered Reload with Prioritization and Frequency Reload Algorithms: this phase is the same in both ordered reload with prioritization and frequency reload algorithms. In the beginning of this phase, XX(I) concurrent entities are generated as in the ordered reload algorithm. Each entity is used to represent one reload process for a disk. Cylinder 1s on all disks are reloaded. After this is done, one more concurrent entity is generated to copy the committed AFIMs from the log to the shadow as described above. In parallel, all reload processes try to reload data needed by waiting transactions into main memory on a cylinder basis. Array 38, which keeps tracks of which cylinders are needed by waiting transactions, is consulted during this process. If a reload process gets to the end of this array, the reload of data needed by waiting transactions for the associated disk is finished. If the reload threshold is not yet reached, the reload process for the mentioned disk then goes back to cylinder 2 on that disk and starts reloading the rest of the cylinders (unneeded cylinders). Every time a cylinder is reloaded, its location in array 38 is set to 2.0. Before reloading a cylinder, this array is examined to see if the cylinder is already reloaded. If it is, then the next cylinder is attempted. Note that in this phase, the average seek is used throughout the reload processes in computing the time the I/O channels must locate the cylinders from appropriate AM disks.

The RP queue that is used to initialize I/Os for unneeded cylinders has a lower priority than the one used for reload of data needed by waiting transactions. Note that when one reload process still reloads cylinders needed by waiting transactions on its disk, other reload processes might have already finished reloading these types of data on their disks, and might be reloading data of a lower reload priority. After a reload process reloads one cylinder on its disk, it checks to see if all cylinders on the disk are already reloaded. If yes, the reload process is terminated. If no, the reload process continues reloading next cylinders.

The reload threshold is calculated by multiplying the reload threshold percentage with the number of disks used XX(I) and the number of cylinders to be reloaded on each disk XX(43). When the reload threshold is reached, the reload threshold is changed to be equal to the database size. The system is brought up. At this point, one reload process is chosen to be split into a number of concurrent entities: one entity to continue the reload process for the associated disk, one entity to represent the special recovery transaction, XX(7)-1 entities to represent normal transactions, and one entity to represent the checkpoint process that will start after a delay of 10 units of time. The rest of the reload processes for the other disks continue their reloading. The transaction entities are then routed to the transaction processing component. The checkpoint entity is branched to the checkpoint component. Each of the reload entities is routed to the reload due to prefetch component where a check to see if more cylinders on its disk need to be reloaded is done.

• Smart Reload Algorithm: in this phase, cylinder 1 which is assumed to store system information on each disk must be reloaded entirely using the cylinder approach. Array 38 locations are now used to keep track of which pages are needed by waiting transactions and which pages are already reloaded. Recall that in the other three algorithms, each location in this array corresponds to one cylinder instead of one page. Similar to the implementation of this phase in the other algorithms, after cylinder 1s on all disks are reloaded, one more entity is created to copy the committed AFIMs from the log to the shadow memory in parallel to the reload processes. Each reload process now starts reloading the page that has the highest access frequency among pages needed by waiting transactions on each disk. To do this, each reload process must consult both array 38 to find out which pages needed by waiting transactions and the priority queue associated with its disk to get the page of the highest access frequency. The RP then requests an I/O for this page and the I/O channel is responsible for reading it from the appropriate AM disk and writing it onto MM. When this is done, the page location in array 38 is set to 2.0 to indicate that the page is already reloaded.

After the reloading of the pages needed by waiting transactions, if the reload threshold is not yet reached, then the reload process is routed back to cylinder 2 and starts reloading unneeded pages with a lower RP allocation priority. When reloading unneeded pages, the reload process on each disk must always consult its associated priority queue and array row 38 to find out

the page that has the highest access frequency among unneeded pages for reload.

The reload threshold is computed in terms of pages. It is equal to the reload threshold percentage multiplied with the number of disks used XX(I) multiplied with the number of cylinders to be reloaded per disk XX(43) multiplied with the number of blocks per cylinder XX(43). When the reload threshold is reached, the system is brought up the same way as implemented in the ordered reload with prioritization and frequency reload algorithms.

Reload Due to Page Faults Phase

This phase takes place when the system is brought up and a page fault occurs. Page faults can be caused by committing the special recovery transaction or by executing waiting or new transactions. Note that this phase is needed in all algorithms but the ordered reload algorithm. Since the reload granularity in the ordered reload with prioritization and frequency reload algorithms is a cylinder while in the smart reload algorithm is a page, the implementation of this phase in the first two algorithms is slightly different from the one in the last algorithm. The descriptions of this phase in these algorithms are given below.

• Ordered Reload with Prioritization and Frequency Reload Algorithms: when a page fault occurs, the DP signals the RP to bring in the needed page. A FORTRAN event is called to identify the disk address which includes the disk number and cylinder number of the wanted page. Since the reload granularity is a cylinder, the cylinder on which the wanted page resides is called the demanded cylinder. The transaction (entity) that causes the page fault is called the current entity. The location in array 38 that corresponds to the demanded cylinder is checked to see if the cylinder has just been brought into MM while DP was signaling RP and while the disk address identification process was going on. If yes, then no reload is needed. The transaction (or entity) that caused the page fault is routed to where its page fault was detected. In doing this, the transaction must be checked to see if it is a special recovery transaction or a normal transaction. The origin where a page fault occurs is different between these two types of transactions. If no, array row 39 is checked to see if the demanded cylinder is currently engaged in a reload process due to a page fault reload process or due to a prefetch reload process. Note that array row 39 is used to keep track of the in-reload-process head position. Array location (39,i) is 1.0 if cylinder i (or page i in the smart algorithm) is currently in the reload process due to page faults; 2.0 if cylinder i is currently in the reload process due to prefetch; and 0.0 if it is not currently in the reload process. If the demanded cylinder is found to be currently engaged in either a prefetch reload or page fault reload, then no reload is performed, instead, the current entity simply waits for the current reload process to finish and is routed back to where its page fault was detected. If none of the above is true, then the current entity is checked to see if it is the special recovery transaction or a normal transaction. One of the following two cases is found:

1) *The current entity (demanding transaction) is the special recovery transaction:* recall that the RP has four different allocation priorities for reload. Each priority is associated with one queue which is called a reload queue. Queue 1 has the highest allocation priority, while queue 4 has the lowest one. The RP will initialize I/O requests for entities waiting in queue 1 before initialize I/O requests for entities waiting in queue 2 and so on. Recall that queue 1 contains the special recovery transaction when this transaction is waiting for the RP to reload its demanded cylinder. Queue 2 contains normal transactions that are waiting for the RP to reload their demanded cylinders. Queue 3 contains reload entities that are waiting for the RP to reload cylinders needed by waiting transactions in the prefetched fashion. Queue 4 contains reload entities that are waiting for the RP to reload unneeded cylinders in the prefetched fashion.

In this case, the four reload queues associated with RP are examined. Queue 1 is checked to see if it contains entities that need to reload the same cylinder as the demanded cylinder. If yes, the current entity is put into a temporary file number 93 and is terminated from the simulation at this time. If no, queue 2 is checked for the same things. If queue 2 contains some entities that need the same cylinder as the demanded cylinder, this means that this cylinder has been requested by transactions of a lower reload priority, the current entity is put into queue 1, and the entities found in queue 2 are removed from queue 2. Since these entities need to continue their processing when the demanded cylinder is reloaded, they are kept in a temporary file number 94 to wait for the reload of the demanded cylinder. If queue 2 does not contain the entities described above, queue 3 is checked. If an entity that tries to reload the same demanded cylinder is in queue 3 the entity is removed from the queue. The current entity is then put into queue 1. If this kind of entity does not exist in queue 3, queue 4 is checked for it. If found, the same action as for queue 3 is done. If not found, the current entity is simply put in queue 1. When the demanded cylinder is reloaded, the temporary files 93 and 94 are searched for the entities that need the same cylinder. These entities are removed from these files and routed back to where their page faults were detected to continue their processing.

2) *The current entity (demanding transaction) is a normal transaction:* a normal transaction can be either a waiting transaction or a new transaction. The four reload queues are also examined as discussed above. If an entity that needs the same cylinder as the demanded cylinder is

found in queus 1 which has a higher RP allocation priority than the current entity, this current entity is put into the temporary file 94 and is terminated from the simulation for now. If the same entity is found in queue 2, the current entity is put into the temporary file 95 and is also terminated from the simulation. If the same kind of entity is found in queue 3 or queue 4, this entity is released from the queue, and routed back to the reload due to prefetch phase. The current entity is then put into queue 2. Note that reload queue 3 contains entities waiting for RP to reload cylinders needed by waiting transactions. Reload queue 4 contains entities waiting for RP to reload unneeded cylinders. It is in the reload due to prefetch phase that these two types of prefetch reload are originated. When the demanded cylinder is completely reloaded, files 94 and 95 are searched for entities that need to reload the same demanded cylinder. These entities are removed from these files and routed back to where their page faults were detected so that they can continue their processing.

- Smart Reload Algorithm: the reload due to page fault phase in this algorithm is very similar to the one described above for the ordered reload with prioritization and frequency reload algorithms, except that in this case, the reload granularity is a page, not a cylinder. The same implementation is applied to the smart reload algorithm with a cylinder being changed to a page.

Model Assumptions/Restrictions

Due to the limitation on the virtual memory assigned to each user on the SMU's IBM 3081, the number of pages used in this simulation cannot exceed 5400. At most 5 AM disks are examined in this simulation. The MM size is assumed to be large enough to store the entire database. The SM size is unlimited.

Simulation Results

We performed eleven testing sets for eleven different dynamic parameters: number of pages, number of disks, read and write probabilities, percentage of transactions committed before the system goes down, multiprogramming level, total number of transactions examined, reload threshold, page size, mean of the exponential distribution for page usage, uniform distribution type for page usage, and number of transactions generated for frequency collection. Due to space limitation, in this section we highlight only the results obtained from running some of these testing sets. Complete results can be found in [Gruenwald, 1990].

Vary Number of Pages

The number of pages is varied between 600 to 5400. Due to the limit on the virtual memory space allocated for each user on SMU's IBM 3081, a number of pages that is larger than 5400 cannot be tested. This does not make the results reported here invalid. In fact, the results obtained show that a conclusion can be derived even before 5400 pages are tested

Figure 4 shows that the reload time is increased as the number of pages increases. This is expected since the database size grows as the number of pages grows. The smart algorithm yields the highest reload time since it reloads data based on the block granularity approach which requires a lot of arm movements to locate a desired block on each disk. The other three algorithms yield very compatible reload time because they follow the cylinder approach in which the seek time and latency are much less than in the block approach. The ordered reload algorithm gives the best reload time as expected. The frequency algorithm, due to its frequency calculation overhead, yields a slightly higher reload time than the ordered reload with prioritization algorithm. On the average, the reload times incurred in the ordered reload with prioritization, frequency, and smart reload algorithms are 4%, 5.5%, and 182%, respectively, higher than the one in the ordered algorithm.

From Figure 5, we see that when varying the number of pages examined, the system throughput incurred by the frequency algorithm remains the best, while the smart algorithm is the worst. In the middle falls the throughput obtained from the other two algorithms. As the number of pages increases, the throughput decreases. The throughput incurred in the smart, ordered reload with prioritization, and ordered reload algorithms are 54%, 10% and 9% lower than that in the frequency algorithm, respectively, when testing with 600 pages, and are 59%, 38%, and 37% lower when testing with 5400 pages.

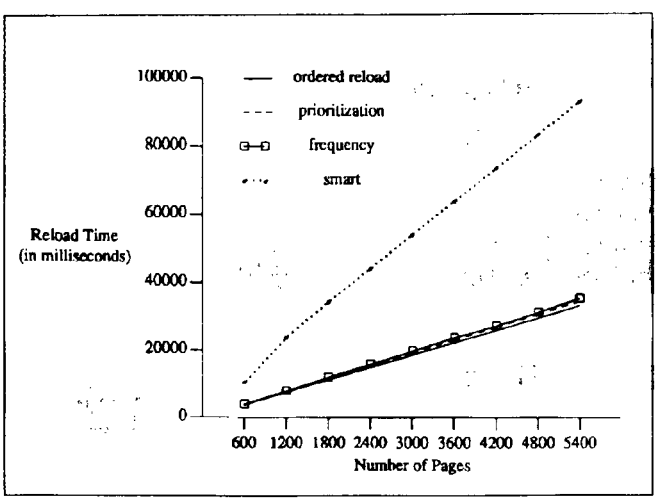


Figure 4. Effects of number of pages on reload time.

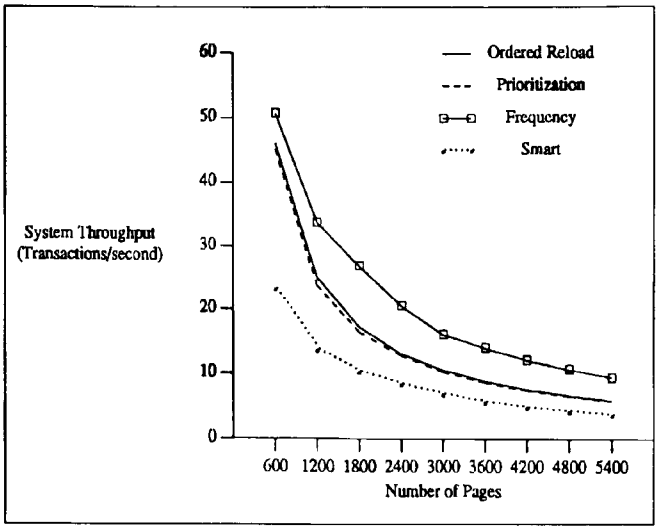


Figure 5. Effects of number of pages on system throughput

Vary Number of Transactions Examined

The total number of transactions examined is changed from 20 to 100. Since the knowledge of data referencing behavior is assumed to be accurate, in each run of this testing set, the frequency information is collected based on those transactions that are examined only. Examining the system throughput shown in Figure 6, the frequency reload algorithm always gives the best result, and the smart algorithm gives the worst result in this testing set. The other two algorithms yield very similar system throughput.

Effects of Inaccurate Prediction of Reference Behavior

The purpose of this experiment is to measure the performance of the algorithms in case the prediction of reference behavior of transactions is not accurate and the hot spots do not exist at all. In this simulation run, a uniform distribution function is used to generate pages for frequency collection while transactions that are created for execution have their pages generated using an exponential distribution function with a random stream that is different from the one used in the uniform distribution for frequency collection. This is enforced to make sure that the frequency collection process is totally independent of the transactions that are examined in the simulation. Table 4 shows the performance of the frequency and ordered reload algorithms in this testing case where time is measured in milliseconds, and system throughput is the number of transactions committed per second.

Table 4 shows that the frequency algorithm gives worse transaction response time and system throughput than those in the ordered reload algorithm. This is due to the fact that the former incurs a high number of page faults because of a lack of hot spots. However, the table also shows that the differences in the reload time, transaction response time, and system throughput are rather small: 4%, 6%, and 4%, respectively.

Summary and Conclusions

Four reload algorithms to recover main memory databases from a system failure have been proposed: ordered reload, ordered reload with prioritization, smart reload, and frequency reload. The differences between these algorithms lie in the structure of the archive memory, utilization of hot spots, reload granularity, reload prioritization, and when to bring the system up. Using the SLAM II language, we have constructed four simulation models to measure the performance of these algorithms. Each model consists of four components initialization, transaction processing, checkpointing, and reloading, each of which has been described in detail in this paper.

The simulation experiments showed that the smart algorithm always gives the worst reload time, system unavailability, number of page faults, transaction response time, as well as system throughput. This fits intuition due to many arm movements that this algorithm must perform to locate a desired block for reload.

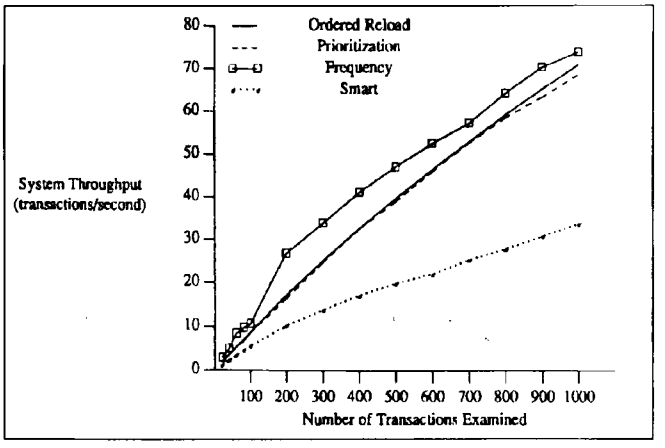


Figure 6. Effects of number of transactions examined on system throughput

Measurements	Frequency Reload	Ordered Reload
Reload Time	11500	11100
Page Faults	45	0
System Unavailability	6880	11100
Transaction Response Time	561	531
System Throughput	16.4	17.1

Table 4. Frequency vs. Ordered when there are no hot spots and frequency collection is independent of transactions executed

The other algorithms give very comparable reload time as intuitively expected since they all use cylinder to be their reload granularity. This result reconfirms our theoretical analysis [Gruenwald, 1990].

The prioritization algorithm performs more poorly than the ordered reload algorithm in terms of transaction response time and system throughput. This is counter-intuitive since the former does not require the entire database to be completely reloaded before bringing the system up while the latter does. Apparently the overhead required to identify waiting transactions and their needed data, as well as the number of page faults incurred in the prioritization algorithm, are too high for the algorithm to yield a positive effect, unless data on AM is organized in access frequency order as we have seen in the frequency reload algorithm.

The frequency algorithm almost always gives the best transaction response time and system throughput in all the testing cases. Its performance gain in the best case outweighs the loss in the worst case. Comparing this algorithm with the second best algorithm, ordered reload, we found that in the best case the frequency algorithm yields 56% more system throughput while in the worst case it yields only 4% less system throughput. The improvement in this algorithm fits intuition due to the fact that hot spots have been taken into consideration, data has been arranged on AM based on access frequency, and the system can be brought online before the entire database is memory-resident. The frequency reload algorithm is thus the chosen one.

References

[1] Ammann, A., Harahan, M., Krishnamurthy, R. "Design of a Memory Resident DBMS",*Proceedings of the IEEE Spring Computer Conference*, 1985, pp. 54-57.

[2] Chou, H., DeWitt, D., "An Evaluation of Buffer Management Strategies for Relational Database Systems", *Proceedings of the International Conference on Very Large Data Bases*, 1985, pp. 127-141.

[3] *Electrical Power Conditioning and Purification Bulletin* 256789-85, Controlled Power Company, 1987.

[4] Copeland, G., Khoshafian, S., Smith, M., Valduries, P., "Buffering Schemes For Permanent Data, *IEEE Transactions on Data Engineering*, 1986, pp. 214-221.

[5] Chris H. Corti and Margaret H. Eich, "Update and Logging Options in Main Memory Databases," May 1991, submitted to *IEEE Transactions on Knowledge and Data Engineering*, under revision.

[6] Deitel, H., *An Introduction to Operating Systems*, Revised First Edition, Addison-Wesley Publishing Company, Chapters 14,15, 1984.

[7] Demurjian, S., Hsiao, D., Marshall, R., *Design Analysis and Performance Evaluation Methodologies for Database Computers*, Prentice-Hall, Inc., 1987.

[8] DeWitt, D., Katz, R., Olken, F., Shapiro, L., Stonebraker, M., Wood, D. "Implementation Techniques for Main Memory Database Systems", *Proceedings of the 1984 SIGMOD Conference*, June 1984, pp. 1-8.

[9] Eich, M. "MARS: The Design of a Main Memory Database Machine", *Proceedings of the International Workshop on Database Machine*, October. 1987, pp. 468-481.

[10] C. Fan, W. Sun, M. Eich, and M. Tanik, "Simulation of a Main Memory Database Based on the Wisconsin Benchmarks," *Southern Methodis~ University Technical Report* 88-CSE-5, January 1988.

[11] Garcia-Molina, H., Lipton, R., Honeyman, P., "A Massive Memory Database System", Princeton University, *Department of Electrical Engineering and Computer Science, Technical Report* September 1983.

[12] Gawlick, D., "Processing Hot Spots in High Performance Systems", *Proceedings of the IEEE Spring Computer Conference*, 1985, pp. 249-251.

[13] Le Gruenwald, "Reload in a Main Memory Database System: MARS", PhD Dissertation Department of Computer Science and Engineering Southern Methodist University, Dec. 1990.

[14] Le Gruenwald and Margaret H. Eich, "MMDB Reload Algorithms," *Proceedings of SIGMOD Conference*, May 1991.

[15] Hagmann, R. "A Crash Recovery Scheme for a Memory Resident Database System", *IEEE Transactions on Computers*, Vol. C-35, No. 9, Sept. 1986, pp. 839-843.

[16] "IBM 3380 Models A04, AA4, and B04, *Direct Access Storage, Description and User's Guide", 4th Edition*, Publication Number GA26- 1664-3, File Number S/370-07,4300-07, IBM, 1984.

[17] Vijay Kumar and Albert Berger, "Performance Measurement of Some Main Memory Database Recovery Algorithms," *Proceedings of the Seventh International Conference on Data Engineering*, April 1991.

[18] Toby Lehman, *Design and Performance Evaluation of a Main Memory Relational Database System*, PhD Dissertation University of Wisconsin-Madison, August 1986.

[19] Lehman, T., Carey, M. "A Recovery Algorithm for a High- Performance Memory Resident Database System", *Proceedings of SIGMOD Conference*, 1987.

[20] Peterson, J., Silberschatt, A., *Operating System Concepts*, Addison-Wesley Publishing Company, 1986.

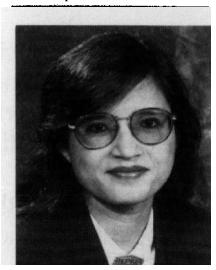
[21] Sacco, M., Schkolnick, M. "A Mechanism for Managing the Buffer Pool in a Relational Database System Using the Hot Set Modeling," *Proceedings of the Conference on Very Large Data Bases, Mexico 1982.*

[22] Sacco, M., Schkolnick, M. "Buffer Management in Relational Database Systems", *ACM Transactions on Database Systems*, Vol. 11, No.4, Dec. 1986, pp. 473498.

[23] Ken Salem and Hector Garcia-Molina, "Disk Striping," *Proceedings of the IEEE International Conference on Data Engineering*, 1986, pp. 33~342.

[24] Salem, K. "Crash Recovery for Memory-Resident Databases", *Princeton University, Technical Report CS-TR-119-87*,November 1987.

[25] Wilkinson, W., "Database Concurrency Control and Recovery in Local Broadcast Networks," University of Wisconsin-Madison, PhD Dissertation, 1981.



LE GRUENWALD is an assistant professor in the School of Computer Science at the University of Oklahoma. She received her BS in Physics from the University of Saigon, Vietnam in 1978, MS in Computer Science from the University of Houston in 1983, and Ph.D. in Computer Science from Southern Methodist University in 1990. She has worked for WRT as a Software Engineer, Southern Methodist University as a faculty member in the Computer Science and Engineering Department, and NEC America, Advanced Switching Laboratory as a member of the Technical Staff in the Database Management Group. Her research interests include Main Memory Databases, Real-Time Databases, Object-Oriented Databases, Distributed Databases, and Expert Database Systems. She is a member of ACM, SIGMOD, and IEEE Computer Society.

MARGARET H. EICH received the B.A. and M.S. Degrees in mathematics from Miami University, Oxford, Ohio and a Ph.D. degree in computer science from Southern Methodist University in 1970, 1972 and 1984 respectively. From August 1984 to the present, she has been first an assistant professor and now associate professor in the Department of Computer Science and Engineering at Southern Methodist University in Dallas. Dr. Eich is currently Associate Chairman of the department. She has published over sixty technical papers in such research areas as database concurrency control and recovery, database machines and main memory databases. Her research has been funded by DOD, JSF, the state of Texas, Texas Instruments Inc., and NEC America Inc. Dr. Eich served as editor of the ACM SIGMOD Record from 1986 to 1988. She has served on the program and organizing committees for several ACM and IEEE conferences, and served as guest editor for a special section of IEEE Transactions on Knowledge and Data Engineering devoted to Main Memory Databases.