MMDB RELOAD ALGORITHMS †

Le Gruenwald
School of Electrical Engineering and Computer Science
The University of Oklahoma
Norman, Oklahoma 73019

Margaret H Eich
Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275

Abstract: In a main memory database (MMDB), the primary copy of the database may be stored in a volatile memory When a crash occurs, a reload of the database from archive memory to main memory must be performed. It is essential that an efficient reload scheme be used to ensure that the expectations of high performance database systems are met. This implies that the overall performance measures of any potential reload algorithm should not be measured simply by reload time, but by its impact on overall system performance. This paper presents four different reload algorithms that aim at fast response time of transactions and high throughput of the overall system. Simulation studies comparing the algorithms indicate that the best overall approach is one based on frequency of access.

1. Introduction

In a *main memory database (MMDB) system,* all or a major portion of the database can be placed in main memory [Eich,1989]. The need for I/O operations to perform database applications is eliminated. With memory costs decreasing and demand for high performance systems rising, MMDBs become an attractive alternative for database systems and have drawn considerable attention from many researchers ([Ammann,1985], [Corti,1990], [DeWitt,1984], [Garcia-Molina,1984], [Hagmann,1986], [Lehman,1987], [Salem,1990], [Son,1989]).

The MMDB model assumed throughout this paper is shown in Figure 1 and is the basis of the *MARS(MAin memory Recoverable database with Stable log)* MMDB system design [Eich, 1987]. It assumes that the primary copy of the database is in a nonvolatile main memory (MM) and that an *archive database* existing on secondary storage (AM) is used solely as a backup in the event of main memory media failure or system failure. A log is assumed to exist both on disk and in a nonvolatile memory buffer All updates take place in a nonvolatile memory (SM) which acts as a shadow memory.

† This material is based in part upon work supported by the Texas Advanced Research Program (Advanced Technology Program) under Grant No. 2265

At commit time, after-image records (AFIM) are copied from the shadow memory to MM. The use of this *shadow-copy* [Salem,1989] updating approach has been validated by previous performance studies [Corti,1990] A database processor (DP) is used to handle normal database processing, while a recovery processor (RP) is used to perform recovery processing activities.

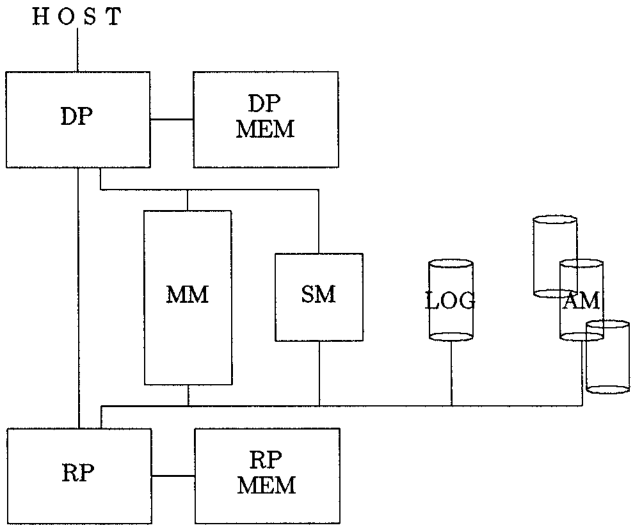

Figure 1 MMDB Model

The AM structure is designed using ***disk striping*** [Salem,1986]. Each disk is divided into two areas: system data cylinders and user data cylinders The system data cylinders consist of data pertaining to system information, such as data dictionary, and address translation tables, while the user data cylinders contain the user data only. The system data cylinders start from cylinder 1 on each disk Following these cylinders are user data cylinders

In an MMDB environment with transaction throughput rates approaching 1000 transactions per second [Gray,1984], performing MMDB reload efficiently is crucial If a system is down for only 1/2 hour then 1,800,000 transactions may be lost To reduce this down time, if the needed data is known, then the database can be brought online before all the database is reloaded According to the

80-20 rule, 80% of database accesses go to 20% of the data [Gray, 1986]. In other words, it is often the case that a transaction can run with only a small portion of the database present in main memory Therefore, it does not make sense to load the entire AM into MM before bringing the database online. These two observations are the motivation for examining the MMDB reload algorithms defined in this paper.

Our objective is to develop an MMDB reload scheme that requires few I/Os, resumes transaction processing quickly and does not degrade the system performance. In Section 2 four different reload algorithms are introduced. Section 3 describes results of simulation experiments comparing the algorithms. The paper is closed with conclusions in Section 4.

2. Reload Algorithms

We propose four different algorithms to perform the MMDB reload process: *ordered reload, ordered reload with prioritization, smart reload,* and *frequency reload.* In the *ordered reload* algorithm, data is reloaded according to the order it is stored on the AM to achieve the fastest reload time. The system is brought up when the entire database is reloaded. This algorithm is examined as it represents the fastest possible reload time but not necessarily the highest overall system performance in terms of MMDB throughput rates. The remaining three algorithms bring the database online prior to the complete reload of MM. They attempt to predict what pages will be needed. Use of these algorithms require that normal transaction processing be able to perform demand paging (page faults) of needed pages in the event that the pages which were prefetched were not sufficient. In *ordered reload with prioritization,* data is reloaded according to a previously determined priority order and the system resumes its execution when a certain amount (percentage) of the database is memory-resident. The *smart reload* algorithm also uses a priority policy to reload the data; however, when data is not reloaded on a demand basis but on a prefetched basis, the block of the highest access frequency is searched and brought into MM. The *frequency reload* algorithm is similar to the *ordered reload with prioritization* except that it requires a special AM structure. In this structure, data is stored according to the order of frequency of access so that data of higher access is reloaded before data of lower access. Each algorithm is further discussed in the following subsections. A more thorough discussion can be found elsewhere [Gruenwald,1990]

Reload granularity is the smallest unit of data to be reloaded from AM into MM. During the reload of this unit, no preemption by a reload of higher priority can take place. The objective is to reload efficiently without degrading the transaction performance. The desirable reload granularity, therefore, should give the best overall system performance in terms of these two properties. Based on earlier performance studies [Gruenwald,1990], cylinder granularity is assumed for all but the smart reload.

2.1. 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.
- Step 2 (performed by RP in parallel with step 1): Copy to the shadow memory all AFIM records of committed transactions on the log.
- Step 3: Bring the system up.
- Step 4 (performed in parallel with transaction processing): Copy the AFIM records mentioned in step 2 from the shadow memory to MM.

Notice that in step 4, the application of AFIMs from the shadow to pages in MM can be performed after the system is brought online. This points out an advantage for the shadow-copy approach 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 [Eich,1987] Thus for recovery, the AFIM from the log need not be applied to the checkpointed pages found in AM prior to bringing the system online

With this algorithm, the database is completely reloaded before the system is brought online. While this yields the fastest reload time it also requires that all data be recovered even if there are no transactions which will use it. Other advantages are that no page faults occur when the system is brought online and it is simple to implement.

2.2. Ordered Reload with Prioritization

This algorithm does not consider access frequency, but does consider reload prioritization and preemption. Its goal is to first reload data that is needed immediately so that the system can be brought up before the entire database is reloaded. Thus 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.
- 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 on the log
- 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 (restart) threshold is reached, the system resumes its execution and priority 1 ensues. 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. Since a cylinder reload granularity is used, reload preemption will not take place until the entire cylinder which is being reloaded is brought into MM.

This algorithm has several advantages over the ordered reload. Only a portion of the database needs to be reloaded before the system can resume its execution. Reload prioritization and reload preemption are taken into account which allow executing transactions to be given immediate attention. The response time of waiting transactions is reduced since data needed by them are reloaded before data that is not needed by any transactions. However, there are still disadvantages. This algorithm is more complicated and requires use of reload prioritization and reload preemption. Identification of waiting transactions and their needed pages adds an overhead to the reload process

2.3. 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], [Gawlick,1985], [Stalin,1990]). A ***hot set (or hot spot)*** is a subset of the database that is frequently accessed. A hot set is a set of pages that have been accessed most frequently in all history of transaction processing.

To reload in the precise order of frequency of access, this algorithm uses a block (page) as the reload granularity. 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.

• Step 4 (performed by RP in parallel with step 3): Copy to the shadow memory the AFIM records of committed transactions on the log.

• 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.

This algorithm has all the advantages of the *ordered reload with prioritization* algorithm. In addition, the block granularity reduces the waiting time for transactions and the more frequently accessed pages are reloaded before the less frequently accessed pages There is extra overhead added to transaction processing due to frequency count calculation. We assume that a frequency counter is associated with each page and that it is updated each time the page is accessed. Since accessing a page requires address translation, this calculation can be implemented as part of the hardware address translation scheme. The block reload granularity requires more seek time and latency time than the one incurred in the cylinder reload granularity Thus the total reload time will be higher.

2.4. 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 increase reload performance. With the frequency AM structure, user data cylinders are arranged based on the decreasing order of page access frequency. Striping of data across disks is still used, with the most accessed data stored on track 1 of all disks, the next highest frequency of access is stored on all tracks 2, etc. Once data is placed on the disks in the correct order, the ***frequency*** reload algorithm works the same as the *ordered reload with prioritization* algorithm

This algorithm is exactly the same as the *ordered reload with prioritization*, except that it is applied on the frequency AM structure, and it assumes that the frequency count information is available in the AM directory This algorithm has all the advantages of the *ordered reload with prioritization* plus it takes advantage of ***hot spots*** and also of minimal seek time and latency time of cylinder reload granularity. However, the computation of access frequency adds an overhead to the reload process and to transaction processing. More nonvolatile memory is used to store the frequency information. Reorganization of the AM structure needs to take place at some point in time since the frequency information does not always remain the same even though the environment is assumed to be nonvolatile. We assume that reorganization takes place when a backup for the AM is created. The backup copy is formed by copying pages from the current AM to a new AM and inserting the pages in the correct locations on the new AM to meet the requirement of a frequency AM structure When the backup process is completed, the RP will switch to the backup AM to perform any subsequent checkpoint. If a system crash occurs at this point, the backup AM is used to reload the data into MM

3. Performance Analysis

In this section we report on a series of simulation experiments performed to compare the four reload algorithms. All experiments were performed on an IBM 3081 using the SLAM II simulation language. The objective is not to reload the database the fastest, but to obtain the best overall performance as measured by transaction throughput and response time.

3.1. Model Description

Figure 2 shows a general diagram of the simulation model The model contains four components. initialization, transaction processing, checkpointing, and reloading. In the initialization 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. During the operation processing phase, each operation of a transaction is processed until completion (no abnormal aborts are assumed to exist). The DP preprocesses an operation and performs SM (Shadow Memory) and MM (Main Memory) address translations in parallel for the page on which the operation is performed. Note that when applied to the frequency and smart algorithms, this phase also incurs a frequency calculation overhead. In the transaction commit phase, the RP performs transaction committing and logging activities. When the number of transactions committed is equal to the number of transactions committed before the system failure parameter, the system is brought down to simulate a system failure The reload component then takes place at this point. In the reload component, there are three different phases initial work upon a system failure, reloading due to prefetch, and reloading due to page faults.

Each transaction is represented as an entity in SLAM II and consists of a transaction identifier, a multiprogramming number, the transaction creation time, and a number of operations. Each operation within a transaction is described with an operation type (read or write) and the page number being accessed Pages are generated by using either an exponential distribution or uniform distribution which is selected before running the simulation Two-phase locking concurrency control is implemented with page level locking and preclaiming of all resources Fuzzy checkpointing is directed by a bit map which indicates modified pages

Tables 1 and 2 show 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 (1:Exponential, 2:Uniform) This function is either exponential or uniform.

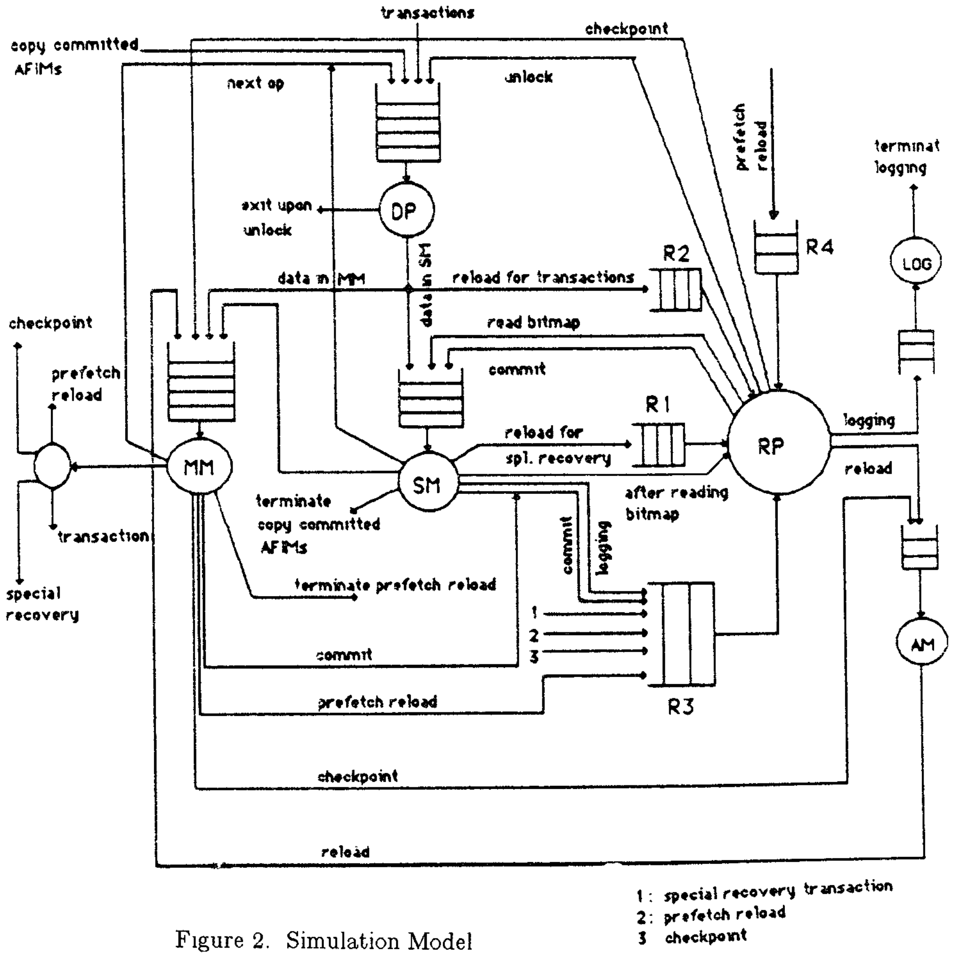

Figure 2. Simulation Model

Parameter	Default Value	Range
Number of AM disks	2	1..5
Number of Pages	1800	calculated
Write Probability	20%	0%..100%
Read Probability	80%	100%-Write Prob
Multiprogramming Level	10	1..20
Total Transactions Examined	200	20..1000
Reload Threshold Percentage	60%	10%..100%
Number of Cylinders per Disk	30	12..60
Blocks per Cylinder	30	15..60
Page Size	23476 bytes	11476..47476 bytes
Transactions Committed Before System Failure	50%	10%..95%
Transfer Time	7.46 ms	3.65..15.09 ms
Random Distribution Type for Page Usage	1	1,2

Table 1 Dynamic Parameters

Parameter	Meaning	Default Value
SM_ACCESS	Access an SM word	0.00011 ms
ALLOC_TM	Allocate a MM page	0.05 ms
AMREQ_TM	Request an I/O from AM	0.02 ms
PRETRAN	Preprocess a transaction	1.25 ms
PREOP	Preprocess an operation	0.005 ms
RELEASE_TM	Release an MM page	0.05 ms
BMAP_TM	Read until 1 in bit map	0.00011 ms
MM_ACCESS	Access an MM word	0.0001 ms
SM_SEAR	AM address translation	0.5*MM_ACCESS
MM_SEAR	MM address translation	3*MM_ACCESS
MSEEK	Minimum seek time	3 ms
REC_SZ	SM or log record	12 bytes
ET_TM	End transaction	1.25 ms
INTIO_TM	Initiate log I/O	0.01 ms
LOGIO_TM	Write a log page	12 ms
LOGPG_SZ	Log page size	2000 bytes
WORD_SZ	Bytes per word	4 bytes
TRACKS_CYL	Tracks per Cylinder	15
SEEK	Average seek time	16 ms
LATENCY	Average latency	8.3 ms
INDN_TM	Initial down time	5 ms
LOCK_TM	Get one lock	0.025 ms
UNLK_TM	Release one lock	0.025 ms
NUM_AFIMS	Number of committed AFIMS in log	10
CPU_POWER	Processor power	2 MIPS

Table 2. Static Parameters

Most of these static parameters are adopted from existing literature. To simulate AM disks, the IBM 3380 disks are used [IBM,1984]. 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 to perform SM address translation is discussed in [Corti,1990]. 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. 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 made to reflect the difference in CPU power. The time to request AM I/O is twice that of the time to initiate log I/O since log access is always sequential while AM access is random based on where the AM record to be accessed is located.

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

- Reload Time. Total elapsed time an algorithm needs to reload the entire database into MM. If the system is brought online prior to completing the reload it will include some time during which transactions are being executed
- System Unavailability. Time during which the system is down
- Page Faults. Number of page faults incurred by each algorithm.
- Transaction Response Time: Mean transaction response time.
- System Throughput: Number of transaction committed per second.

Among these measurements, the last two are the most crucial because they indicate overall system performance.

3.2. Simulation Results

In the following subsections we highlight the results of the simulation experiments [Gruenwald,1990]

3.2.1. Vary Number of Pages

The number of pages is varied between 600 to 5400 In Figure 3, the reload time incurred in each of the reload algorithm as the number of pages is varied is plotted. The smart algorithm yields the highest reload time since it reloads data based on the block granularity approach which may require a number of arm movements to locate a desired block. 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.

An interesting result occurs when examining the effect of the number of page faults incurred by the reload algorithms Figure 4 shows that as the number of pages increases, the number of page faults in the ordered reload with prioritization algorithm also increases. This is due to the fact that this algorithm does not take frequency of access into consideration. As the database size grows, the probability that transactions access the pages beyond the reload threshold also grows. The ordered reload, as expected, incurs no page faults at all because this algorithm must reload the entire database before bringing the system online. The frequency and smart algorithms both experience a drop in the number of page faults. The frequency algorithm effectively has no page faults with 1800 pages in the database while the smart one reaches this state with 3000 pages. Recall that these algorithms continue to reload the database after transaction processing is resumed. Since the most frequently referenced data is reloaded first, when these algorithms are used the background reload processes will have time to bring in data before it is needed. The advantage that the frequency algorithm has over the smart one is that with each page faults more data (cylinder vs. page) is loaded into memory.

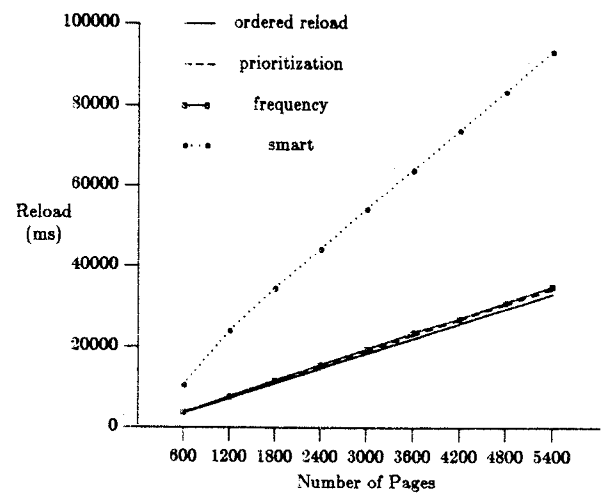

Figure 3. Effect of Number of Pages on Reload Time

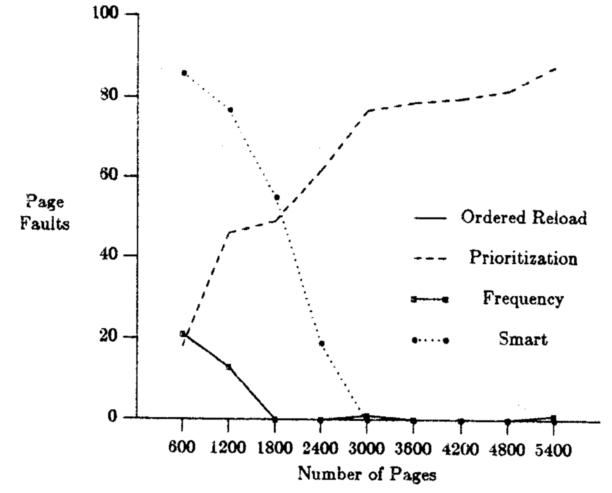

Figure 4. Effect of Number of Pages on Page Faults

The effect of the number of pages on the mean transaction response time is shown in Figure 5. As seen in this figure, the smart reload algorithm gives the highest transaction response time. This is due to the fact that its system unavailability is the highest among the four proposed algorithm. The ordered reload and ordered reload with prioritization algorithms have comparable transaction response time. The ordered reload algorithm, even though it incurs higher system unavailability than the ordered reload with prioritization, has no page faults at all while the latter incurs quite a few page faults. Since the latter uses time to bring in faulted pages, it does not not yield better response time than the plain ordered reload. With the frequency algorithm, fewer page faults are incurred, therefore the response time is lower than in the other algorithms. When the number of pages is small, 600 pages, the response times in the smart, ordered with prioritization, and ordered reload algorithms are respectively 111%, 10%, and 6% higher than that of the frequency reload algorithm. When the number of pages is large, 5400 pages, these figures also become higher: 146%, 60%, and 59%, respectively. Similar results were found when examining transaction throughput.

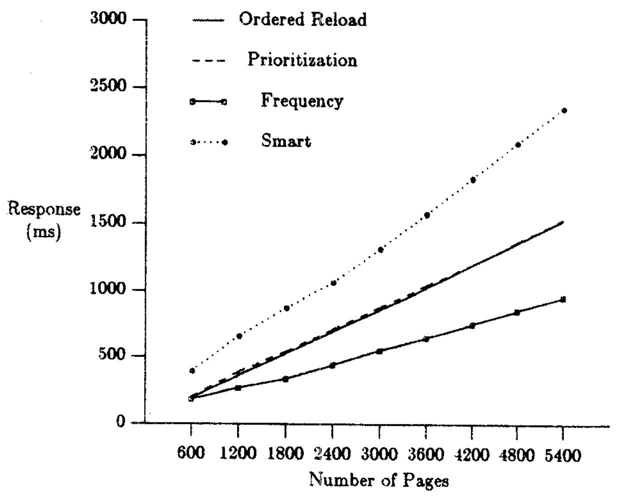

Figure 5. Effect of Number of Pages on Transaction Response Time

3.2.2. Vary Reload Threshold Percentage

The reload threshold percentage dictates the number of pages (as in the smart reload algorithm) or the number of cylinders (as in the ordered reload with prioritization and the frequency reload algorithms) to be reloaded before the system can be brought up. In this testing set, this percentage is varied from 10% to 100%. Note that this testing set is not applied to the ordered reload algorithm since this algorithm always requires the reload threshold percentage to be 100%. As seen in Figure 6, the frequency reload algorithm gives better transaction response time than the smart and the ordered reload with prioritization algorithms. When the reload threshold reaches 100%, the frequency reload and the ordered reload with prioritization algorithms give the same results. As the reload threshold increases, the smart reload algorithm increasingly gives the worst results. Even though, the number of page faults in this algorithm keeps decreasing when the reload threshold increases, the decrease in system unavailability due to later restart outweighs the savings gained in the number of page faults. Similar results were found for transaction throughput.

The ordered reload with prioritization is not very sensitive to the reload threshold while the frequency and smart reload algorithms are quite affected by this parameter. In general, the higher reload threshold used, the higher is system unavailability and, hence, there is less benefit from using access frequency. The worst transaction response time and system throughput are then obtained.

However, when the reload threshold is too small, many page faults might be incurred with the frequency algorithm. This could outweigh the gain in system availability, and in turn degrade transaction response time and system throughput obtained from this algorithm. Therefore, to take advantage of access frequency, the reload threshold must be selected carefully, and should be a small number. below 50%. The loss in system performance when choosing a too small reload threshold is much less than the loss when choosing a too big reload threshold.

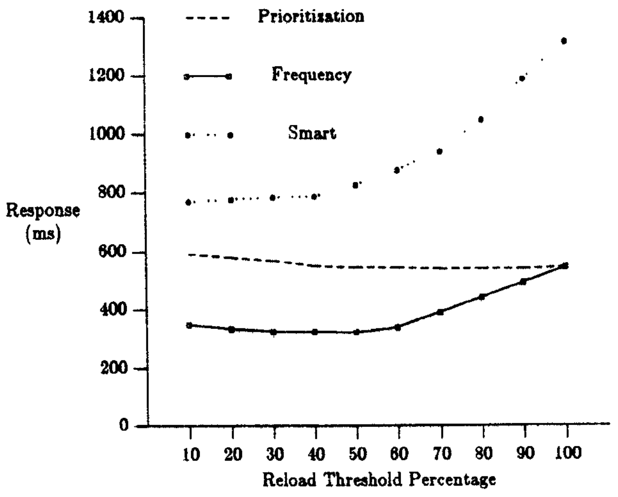

Figure 6. Effect of Reload Threshold Percentage on Transaction Response Time

3.2.3. Vary Number of Transactions

Experiments were performed which varied the number of transactions executed from 20 to 1000. These experiments predict the performance as the mean time between failure increases. As the number of transactions increases the number of page faults in the smart algorithm increases while in the frequency and ordered reload algorithms, this number is relatively constant with the frequency algorithm always being less. When examining transaction response time, we find that the frequency algorithm clearly gives better results when the number of transactions examined is less than 500. From that point on comparable results are found for the frequency, ordered reload, and prioritization algorithms. Examining the system throughput, the frequency reload algorithm always gives the best results, and the smart algorithm the worst.

3.2.4. Suppose No Hot Spots?

In this testing set, pages accessed by transactions are generated using a uniform distribution function. This simulates no hot spots in actual processing. The simulation shows that the smart algorithm gives the highest reload time, highest system unavailability, highest transaction response time, and lowest system throughput. The frequency algorithm gives the lowest transaction response time, and the highest system throughput. The reload time is the same for both frequency and ordered reload with prioritization algorithms. The ordered reload gives the lowest reload time, and no page faults at all. It also gives better transaction response time and system throughput than the ordered reload with prioritization algorithm. The number of page faults incurred in the latter is the highest in all algorithms. This testing also indicates that when changing the distribution function for page usage, the results are very much unchanged from those found with an exponential distribution.

Table 3 shows the performance of the algorithms when generating pages using an exponential distribution function and using a uniform distribution function. Even though the relative order in reload time, transaction response time, and system throughput remains the same across all algorithms, the frequency algorithm performs worse when using the uniform distribution function than when using the exponential distribution. This is expected because the hot spots get referenced more often in the latter case. Reloading of data that is stored on AM disks using the frequency AM structure is therefore more advantageous. The smart algorithm, on the otherhand, when using the uniform distribution gives fewer number of page faults and better transaction response time and throughput. This is due to fact that the hot spots now are distributed more evenly over all disks. The reload of data in strictly decreasing order of frequency in parallel from all disks is therefore advantageous. The table also shows that the distribution has no effect on the ordered reload algorithm and very little effect on the ordered with prioritization. As mentioned earlier, the ordered reload algorithm reloads the entire database before bringing the system up. Hot spots have no effect on it. Therefore, the algorithm is not affected by the distribution function selected for generating pages. The ordered reload with prioritization also does not take frequency of access into consideration, and hence is similarly unaffected by the distribution.

Measurements	Ordered	Prior	Freq	Smart
Unif. Reload	11100	11600	11600	32900
Expo. Reload	11100	11500	11800	34600
Unif. Page Faults	0	50	9	24
Expo. Page Faults	0	49	0	55
Unif. Response	531	561	390	811
Expo. Response	531	545	341	878
Unif. Throughput	17.1	16.3	21	11
Expo. Throughput	17.1	16.3	26.7	10.3

Table 3. Exponential Distribution vs. Uniform Distribution

3.2.5. Effect of Inaccurate Prediction of Reference Behavior

Simulation results reported so far show that the frequency reload algorithm yields the best transaction response time, and system throughput. These results reflect the situation when data referencing behavior is predicted accurately. This is due to the fact that the frequency information was collected only for those pages that are going to be referenced by the examined transactions. What would happen if the prediction is not accurate? To answer this question, another testing set is performed. In this experiment, the number of transactions generated for frequency collection is varied from 200 to 5000, and the number of transactions to be examined in the simulation run remains at its default value of 200.

The effects of the number of transactions generated for frequency collection on transaction response time in the frequency and ordered reload algorithms is shown in Figure 7. The reload time incurred by the frequency algorithm remains higher than the one in the ordered reload algorithm. The difference in the reload time between the two algorithms is about 7%. When more transactions are generated for frequency collection than those examined, more page faults tend to occur. The transaction response time therefore keeps increasing and system throughput keeps decreasing. At some point, when the number of transactions generated for frequency is large enough, for example 1000 in this case, the frequency algorithm and the ordered reload yield comparable performance. The throughput and response time do not change very much after this point even though the number of page faults is still changing.

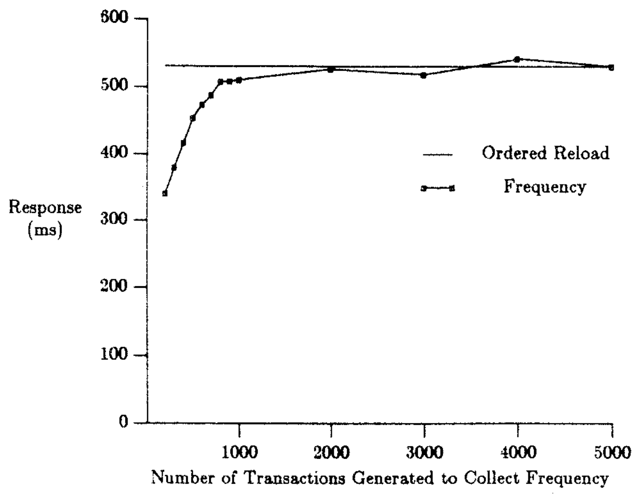

Figure 7. Effects of Transactions Generated to Collect Frequency on Transaction Response Time

The next question posed is what would happen if the prediction about data accessing behavior of transactions has no bearing on the actual access frequency collection and the hot spots do not exist at all. A simulation run is performed to answer this question. In this simulation run, a uniform distribution function is used to generate pages for frequency collection. 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 experiment examines the behavior of the frequency algorithm in the worst case. 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.

Measurements	Frequency	Ordered
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

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 in hot spots. However, the table also shows that the differences in the reload time, transaction response time, and system throughput are extremely small: 4%, 6%, and 4%, respectively. This simulation therefore substantiates the results from the previous testing case that even in the worst case when there is no knowledge of frequency of access, the frequency algorithm is comparable to the ordered reload algorithm.

4. Conclusion

Four reload algorithms have been proposed: ordered reload, ordered reload with prioritization, frequency reload, and smart reload. Simulation results showed that the ordered reload always gives the minimum reload time because there is no processing interfering with its reload process. However, the behavior of ordered reload with prioritization and frequency reload approaches are very close. The difference in reload time across these three algorithms is only about 5%. The smart algorithm, due to many arm movements that it must perform to locate a desired block for reload, yields about 200% higher reload time than the other algorithms. In terms of transaction response time and system throughput, the smart algorithm † always give the worst performance while the frequency almost always give better performance than the other three algorithms. In the best case, when the knowledge of data referencing behavior is accurate, the frequency algorithm yields 36%, 37%, and 61% less transaction response time and 56%, 64%, and 159% higher system throughput than the ordered reload, ordered reload with prioritization, and smart reload algorithms, respectively. When there are no hot spots on the database and transactions to be executed in the simulation are entirely independent of the access frequency collection on hot spots, the frequency algorithm results in only 6% higher transaction response time and 4% lower system throughput than the ordered reload algorithm.

The objective of this research was not to discover the fastest database reload algorithm, but to find a way to reload the database in such a way that the system can resume its execution quickly to achieve high performance in terms of transaction response time and system throughput. The simulation results showed that the frequency algorithm then is the algorithm of choice.

† Perhaps we should have named this the "dumb" not the smart algorithm.

5. References

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

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

[Corti,1990] Chris H Corti and Margaret H Eich, "Update and Logging Options in Main Memory Databases," May 1990 submitted to *IEEE Transactions on Knowledge and Data Engineering.*

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

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

[Eich,1989] Margaret H. Eich, "Main Memory Database Research Directions," *Proceedings of the 1989 International Workshop on Database Machines*, Deauville, France, June 1989, pp. 251-268

[Fan,1988] C Fan, W Sun, M Eich, M Tanik, "Simulation of a Main Memory Database Based on The Wisconsin Benchmarks," Southern Methodist University Technical Report 88-CSE-5, January 1988

[Garcia-Molina,1984] Hector Garcia-Molina, Richard J. Lipton, and Jacobo Valdes, " Massive Memory Machine," *IEEE Transactions on Computers*, Vol C-33, No 5, May 1984, pp 391-399

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

[Gray,1984] J Gray, B. Good, D. Gawlick, P. Homan, and H Sammer, "One Thousand Transactions Per Second," Tandem Computers, Technical Report 85 1, Nov 1984

[Gruenwald,1990] Le Gruenwald, *Reload in a Main Memory Database System: MARS*, PhD Dissertation Department of Computer Science Southern Methodist University, August 1990

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

[IBM,1984] "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

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

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

[Salem,1986] Kenneth Salem and Hector Garcia-Molina, "Disk Striping," *Proceedings of the IEEE International Conference on Data Engineering*, February 5-7, 1986, pp. 336-342

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

[Salem,1989] Kenneth Salem, *Failure Recovery in Memory Resident Transaction Processing Systems*, PhD Dissertation Princeton University, January 1989.

[Salem,1990] Ken Salem and Hector Garcia-Molina, "System M: A Transaction Processing Testbed for Memory Resident Data," *IEEE Transactions on Knowledge and Data Engineering*, Vol 2, No 1, March 1990, pp. 161-172

[Son,1989] S H Son, "Recovery in Main Memory Database Systems for Engineering Design Applications," *Information and Software Technology*, Vol 31, No. 2, March 1989, pp. 85-90.

[Stalin,1990] Carl Stalin and Hector Garcia-Molina, "File System Design Using Large Memories," *Proceedings of the 5th Jerusalem Conference on Information Technology*, October 1990, pp 11-21