A Data Priority Reload Technique for Real-Time Main Memory Databases

Jing Huang Le Gruenwald[1]
School of Computer Science
The University of Oklahoma
200 Felgar Street, Room 114 EL
Norman, OK 73019
EMAIL: gruenwal@mailhost.ecn.uoknor.edu
PHONE: (405) 325-3498
FAX: (405) 325-4044

Abstract

In this paper a data priority reload algorithm is proposed for real-time main memory database systems that allows transaction processing to be resumed before the entire database is recovered from a system failure. Transaction execution priority, reload priority and reload preemption are taken into account during the reload process; this enables transactions with high execution priorities to have more opportunities to meet their deadlines and thus enhances the overall system performance. This algorithm also gives temporal data a higher reload priority than persistent data so that many temporal data can be used before losing their validity, which in turn minimizes the number of transactions aborted or delayed due to invalid data access. Based on the simulation results obtained, we can draw the following conclusions: (1) The data priority reload algorithm provides a significant performance improvement over the conventional reload approach. (2) Whether the efficiency of different reload algorithms is crucial to the overall performance mainly depends on the system load, database size and system failure rate. (3) The key factors of the design of real-time MMDB reload algorithms are system unavailability, transaction execution priority, and reload threshold. In order to derive an efficient reload scheme, these factors must be studied carefully.

1. Introduction

Many real world applications involve time constrained access to data, for example, telephone switching systems, program stock trading and radar tracking. All of these entail gathering data from the environment, processing gathered information, and providing responses within a specified time or deadline. Another aspect of these applications is that they might process both *temporal data*, which lose their validity after a certain time interval, and *persistent data* which remain valid regardless of time. Applications such as these introduce the need for *real-time database systems* (RTDBS). The main goal of an RTDBS is to meet the timing constraints of transactions and data [14]. In order to achieve higher system performance, the use of a main memory database (MMDB) is a good choice. This is because in an MMDB environment, all or a major portion of the database can be memory-resident, thus transaction processing can be satisfied with few I/Os. An MMDB system therefore has a potential for obtaining a substantial performance improvement over a disk-resident database system for real-time applications.

However, due to the volatility of semi-conductor memory, the contents of main memory may be lost at the time of system crash caused by power failures, software or hardware errors. On average, a system failure occurs once a week ([5], [7]). As the database is not available for transactions when a failure occurs, system performance degrades severely. For a large database, simply transferring it from archive disks to main memory can be very time-consuming. As estimated in [11], 28.43 minutes are needed to recover one gigabyte database. For a high performance system which processes 1000 transactions per second, there might be many transactions that are backlogged during system recovery, which may cause many transactions to miss their deadlines and a large amount of

[1] This material is based in part upon work supported by National Science Foundation under Grant No. IRI-9201596.

1068-3070/96 $5.00 © 1996 IEEE
Proceedings of EURWRTS '96

temporal data to lose their validity before they can be used. In order to resume transaction processing quickly without degrading the system performance, an efficient reload technique, which reloads the archive database into main memory and reconstructs the consistent state of the database, needs to be developed.

Several reload techniques have been developed for MMDB database systems [6]; however, not much research has been done for RTDBS. In this paper, we propose a data priority reload technique for a real-time MMDB system, which aims at not only reducing the system restart time, but also minimizing the number of timing constraints which are violated. In order to evaluate the performance of the proposed reload technique, extensive simulation experiments are conducted.

The remainder of this paper is organized as follows. We start in Section 2 with a description of the system assumption used throughout this study, then introduce in Section 3 the reload terminology we will use to discuss the reload algorithms in the following sections. The data priority reload algorithm is presented in Section 4. The simulation model and methodology are provided in Section 5. Performance experiments and results are analyzed in Section 6. Finally, Section 7 concludes the paper.

2. System Assumption

In this section, we introduce the system architecture which will be used throughout this study. Two processors, Database Processor (DP) and Recovery Processor (RP), running in parallel are assumed in the system. The DP handles normal transaction execution while the RP manages transaction termination, logging, checkpointing and recovery from a system failure.

We assume that the entire database is stored in a volatile Main Memory (MM), while its backup copy is kept in an Archive Memory (AM) residing on secondary storage. The database on AM is updated only when a checkpoint is taken. Fuzzy checkpointing [8], which does not require the system to be quiescent during the checkpoint process, is assumed in this work. Based on our previous studies [9], when being combined with deferred update, the logging scheme, which logs both valid and invalid temporal data, and maintains persistent data and temporal data log records in separate log buffers, gives the best performance in terms of logging space, number of memory references and cost to perform REDO operations. This technique is thus assumed in this work. With this logging technique, modified data are kept in the log until a successful completion of the transaction performing the updates is assured, at which time the modifications are applied to the database. Log buffers are assumed to be large enough to contain all updates of active transactions. When a log buffer is full, its contents will be flushed to a log disk.

As we mentioned earlier, temporal data may get old or become invalid if they are not updated within a certain period of time. To quantify this notion of "age", each temporal data item is associated with two attributes: timestamp (T_{update}) and valid interval (*Interval*). T_{update} indicates the update time of a temporal data item, which denotes the real time when an observation relating to the data item is made, and *Interval* represents the absolute validity interval of a data item, i.e., the length of time interval following the update time during which the data item is considered to have absolute validity. A temporal data item is said to be valid or meet the absolute temporal consistent requirement at the current time T_{now} if $Tnow - Tupdate < Interval$ holds.

We also assume that for temporal data whose valid intervals are shorter than the time needed to reload a cylinder into MM, no logging and checkpointing will be performed on them. Note that a cylinder is assumed to be the smallest unit of data to be reloaded without preemption used by the system for prefetch reload. This is because in a real-time database system, there may be some temporal data items whose valid intervals are very short. After a system crash, by the time the AM is accessed for database reload and post-crash log processing is performed for the reloaded database in MM, they would have already become invalid. These will not only result in a waste of system resources such as log buffer space, log disk flush time and recovery time, but may also delay the fresh updates for these data items, which consequently leads to a poor system performance in terms of transaction meeting deadlines.

3. Reload Terminology

This section explains all the terminology we will use in our discussion. *Transaction Priority* represents the execution priority of a transaction, which the resource manager uses to assign resources, such as DP and RP, to the transaction during its execution. *Reload Preemption* means that reload of some data is suspended and replaced by reload of some other data. *Reload Granularity* is the smallest unit of data to be reloaded from AM to MM. During the reload of this unit, no preemption by a reload of higher priority is allowed. *Recovery Unit* is the smallest unit of data to be recovered. The process of recovering a unit includes reloading pages belonging to the unit from AM to MM and processing all the log records associated with it. No access is allowed to a unit during the course of its recovery. *Reload Priority* indicates which data will be reloaded first and which data will be reloaded next. *Re-*

load Threshold specifies the amount of database that must be memory-resident before the system can be brought up.

4. Data Priority Reload Algorithm

The main features of the proposed reload algorithm include 1) the system is brought on-line before the entire database is reloaded into MM in order to reduce down time; 2) transaction execution priorities are taken into account during the reload process to give immediate attention to high priority transactions so that they have more opportunities to meet their deadlines and 3) data accessed frequently are reloaded before other data so that the number of page faults can be reduced and transaction execution can be processed with fewer interruptions. 4) Temporal data are given a higher reload priority than persistent data so that many temporal data can be used before losing their validity; this will reduce the number of transactions aborted due to invalid data access and in turn improve the system performance in terms of transactions and data meeting timing constraints

In the following subsections, we first introduce the AM structure required by the data priority reload algorithm, and then give a detailed description of the algorithm.

4.1. AM Structure

To facilitate the reload process, the *disk striping* technique [2], which distributes data transparently over multiple disks to make them appear as a single fast and large disk, is used. Each disk is divided into two areas: system cylinders and user data cylinders. The system data cylinders start from cylinder 1 on each disk and store all the system information such as data dictionary and address translation table, following these cylinders are user data cylinders in which the backup copy of the MMDB resides. As in the data priority reload algorithm, temporal data are assigned with higher reload priorities than persistent data, and within each data type, pages with high access frequencies are reloaded into MM before those of low access frequencies. To facilitate the reload process, the AM is structured in such a way that temporal data are stored first, and then followed by persistent data. Within each data type, pages are placed based on the decreasing order of 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, and temporal data will be reloaded into MM before persistent data.

4.2. Algorithm

In detail, this algorithm consists of the following steps:

1. Construct the recovery buffer and recovery bit map. This step will be explained later.
2. Reload system pages into MM on a cylinder basis (Performed by RP).
3. Reload the rest of the database based on the following prioritization until the reload threshold is reached:
   a) Priority 1 (higher reload priority): The RP reloads temporal data according to their access frequencies on a cylinder basis. The highest access frequency page is reloaded before the second highest access frequency page and so on. After a page is brought into MM, the DP checks the status (clean or dirty) of the page by using the recovery bit map. A page is considered to be clean if it is not modified since the last complete local checkpoint; otherwise, it is dirty. If a page is found to be dirty, its corresponding log records that are organized in Step 1 are applied to the page to bring it up to its state preceding the crash.
   b) Priority 2 (lower reload priority): The RP reloads persistent data according to their access frequencies on a cylinder basis. The data page which has a higher access frequency will be reloaded into MM before that with a lower access frequency. The status of the reloaded page is checked by DP after it is brought into MM and a recovery action similar to that described in Step 3.a will be taken if the page is dirty.
4. Bring the system up when the reload threshold is reached.
5. Reload the rest of the database based on the following prioritization until the entire database is memory-resident. The status of each page is checked after it is brought into MM, and log records associated with the page are applied to it (Performed by RP):
   a) Demand reload priority (highest priority): Demand reload based on transaction execution priority. When a page fault occurs, the execution of the requesting transaction is suspended until the needed page is brought into MM and log records (if any) associated with this page are applied to it in MM. In this

step, reloading is performed on a page basis.

b) Prefetch reload priority 1: Reload temporal data according to the decreasing order of their access frequencies on a cylinder basis.

c) Prefetch reload priority 2 (lowest priority): Reload persistent data according to the decreasing order of their access frequencies on a cylinder basis.

Note that prefetch reload is performed on a cylinder basis while demand reload is on a page basis. It is desired that database reloading can be finished in the shortest amount of time while at the same time incurring less overhead on normal processing. As pages stored on a cylinder can be reloaded sequentially in a minimal amount of time, cylinder is thus selected to be the reload granularity for prefetch reload. However, when a page fault occurs, in order to bring the requested page into MM as quickly as possible so that the requesting transaction can proceed without too much delay, page reload granularity is used.

Another point needs to be noticed is that in this algorithm, temporal data are given higher reload priorities than persistent data. This can be explained as follows. When a page fault occurs, if it is a persistent data page, the requesting transaction can perform its execution after the needed page is brought into MM. However, if it is a temporal data page, transaction execution may not be able to proceed even though the requested data page is brought into MM. This is because we assume that a transaction will be aborted if a temporal data item which the transaction accesses is out of date. As a temporal data page tends to become invalid after the system is resumed from a failure, if it is not updated immediately, there is a very high chance that it will lose its validity after it is reloaded into MM by the requesting transaction. This will results in an abortion of the requesting transaction. Thus, reloading a certain amount of temporal data into MM before resuming transaction execution might improve the overall system performance.

As transaction execution can proceed when needed pages are memory-resident and recovered, it is possible that in the log buffers, the log information of pages which have not been reloaded will be mixed together with the log information generated by executing transactions which arrive after the system is brought on-line. To avoid an incorrect recovery, the recovery bit map and recovery buffer need to be constructed before reload starts. The recovery bit map indicates whether a page is dirty or has been modified since the last checkpoint and needs to be recovered before being accessed by transactions. Each bit in the recovery bit map corresponds to a page in MMDB. The recovery buffer maintains the corresponding recovery information for the above dirty pages and is stored in the non-volatile memory. In order to efficiently recover each page, log records associated with the same page are grouped together in the recovery buffer.

5. Simulation Model and Methodology

In order to measure the performance of the proposed reload algorithm, we developed a simulation model of a centralized real-time MMDB system and performed extensive experiments. Only firm deadline transactions, which are discarded if they are not completed by their deadlines, are considered in the model. The *earliest deadline policy* [1] is selected for transaction execution priority assignment. This policy gives a transaction that has the earliest deadline the highest execution priority. If two transactions have the same deadline, the transaction which arrives at the system earlier is considered to have a higher execution priority. The "conditional restart" using the 2-phase locking concurrency control mechanism [1] is adopted in our simulation. For simplicity, we assume that when a transaction enters the system, before it can be processed, it must obtain all needed locks on pages it is going to access. At its commit time, the transaction releases all its locks.

The parameters used to specify the system configuration and workload are derived based on the DEC 3000 Model 400/400S AXP Alpha workstations [3] and Micropolis 22000 disk drivers [12], since they accommodate high performance applications. The detailed explanation about parameter settings can be found in [10].The performance metric used here is the percent of transactions missing deadlines after a crash, which is observed from the time at which the system goes down until the end of a simulation run. A reload algorithm (referred to as the *conventional approach* in this paper), which brings the system on-line when the entire database is memory-resident and all the log information is processed, is used to compare with out proposed reload techniques.

There are two types of transactions in the system: aperiodic and periodic. *Aperiodic transactions* or normal transactions are generated using an exponential distribution stream at a specified mean rate. Each normal transaction submitted to the system is associated with its creation time, transaction identifier, transaction size, operations, pages on which operations are performed, deadline and execution priority. *Periodic transactions* are write-only transactions. Each periodic transaction is responsible for updating one temporal data page and is invoked at the beginning of its update period. The time interval during which a periodic transaction is triggered in order to

maintain the absolute temporal consistency is called update period. The update period of a periodic transaction is defined to be half of the corresponding temporal data's valid interval. The deadline of a periodic transaction is assumed to be the end of its update period.

If a normal transaction is found to read an invalid temporal data page, in order to give a chance to the periodic transaction which is responsible for updating the temporal data page, the normal transaction will be aborted and releases all its locks. If the normal transaction still has a feasible deadline, it will be scheduled to restart later; otherwise, it is discarded.

Except in the testing case which examines the effects of the system failure rate on the proposed reload scheme, in all other testing cases a system crash is assumed to take place once at the time when half of the specified normal transactions are finished (committed or aborted). When a system failure occurs, all resources are made inactive, and all active transactions are kept in a file so that they can be restarted later. The MM is emptied to simulate its volatility. Transaction execution is resumed when the reload threshold is reached. The rest of the database is reloaded into MM by demand or prefetch reload in parallel with normal transaction processing. The simulation model is written using the simulation language SLAM II [13]. In each simulation experiment at least 10,000 normal transactions are executed for all the reload algorithms. The final results are obtained by averaging the results over 20 independent runs. 95% confidence levels are obtained for the performance results. The width of the confidence interval of each data point is within 5% of the point estimate.

6. Simulation Results

Totally, five testing cases, varying transaction arrival rate, varying database size, varying reload threshold, varying percent of temporal data and varying system failure rate, are performed. The results obtained are highlighted as follows. The data priority reload algorithm, in most cases, offers a better overall system performance in terms of transactions and data meeting timing constraints than the conventional reload approach.

Whether the efficiency of different reload algorithms is crucial to the overall system performance mainly depends on system load, database size and system failure rate. As shown in Figure 1, the higher the transaction arrival rate is, the more important the performance of the reload algorithm becomes. This is because the number of transactions that are backlogged when a system failure occurs is increased as the arrival rate is increased. If transaction processing cannot be resumed quickly, many transactions and data may miss their timing constraints. An efficient reload algorithm is also desired when the database size becomes large. As for a large database, simply reloading the entire database into MM will take a great amount of time, many transactions may miss their deadlines due to this delay. In this case, the time at which the system operation can be resumed and how efficiently pages requested by high priority transactions can be brought into MM becomes a crucial factor to the overall system performance. The results obtained also show that the higher the system failure rate is , the more important choosing an efficient reload algorithm becomes.

The key factors of the design of a real-time MMDB reload technique are system unavailability, reload priority, transaction execution priority and reload threshold. In order to achieve a good recovery performance, the decrease in system unavailability is more important than the decrease in database reload time. Even though the conventional reload approach can finish database reloading in the shortest amount of time, as transaction processing is delayed for a long time compared with that in the proposed reload scheme, it finally results in the worst performance. To reduce the amount of invalid data and the number of transactions aborted due to invalid data access, temporal data of short valid intervals should be reloaded into MM as quickly as possible. In addition, pages needed by high execution priority transactions should be given higher reload priority so that the execution of these transactions will not be suspended too long. The results obtained indicate that taking transaction's execution priority into account during database reloading can reduce the number of transactions missing deadlines. Reloading most frequently access pages before other data also helps reduce the number of page faults, which in turn hastens transaction processing. To take advantage of data priority reload, the reload threshold must be selected carefully and should not be a small number. As plotted in Figure 2, the loss in the system performance when choosing a too large reload threshold is much less than the loss when choosing a too small reload threshold.

7. Conclusions

In this paper we presented a data priority reload technique for a real-time MMDB system. This algorithm enables database reloading and transaction processing to be performed in parallel, which reduces system unavailability and enhances the overall performance. This algorithm takes transaction execution priority, reload priority and preemption, data characteristics such as temporality and access frequency into account during the reload process. It reduces the amount of invalid temporal data and the number of transactions aborted due to invalid data access and enables higher priority transactions to be given immediate

attention so that they have more chances to meet their deadlines. The simulation results obtained show that 1) the proposed reload algorithm has a potential to offer a better performance than the conventional approach; 2) the selection of an efficient reload becomes more critical under the environment in which the system load is high, the database size is large or the system failure occurs frequently; 3) the key factors to the design of a real-time MMDB reload scheme are system unavailability, transaction execution priority, reload priority and reload threshold.

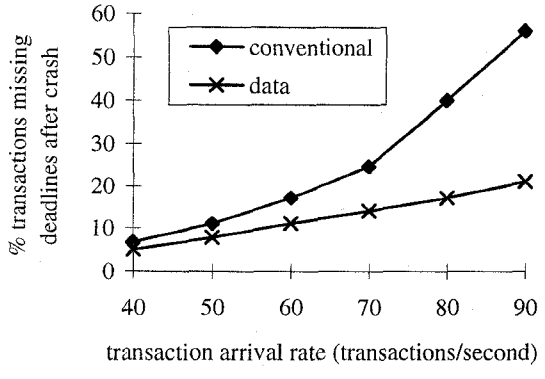

Figure 1. Transaction Arrival Rate vs. % Transactions Missing Deadlines After Crash

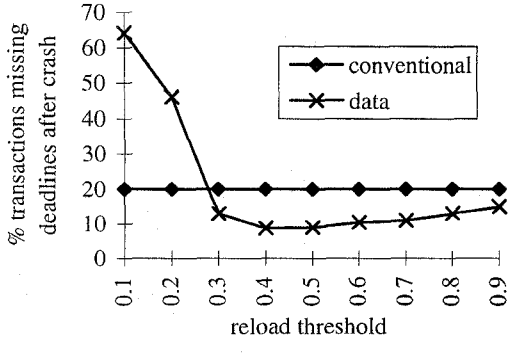

Figure 2. Reload Threshold vs. % Transactions Missing Deadlines After Crash

References

[1] R. Abbott, H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation", ACM Transaction on Database Systems, Vol. 19, No. 3, September 1992, pp. 513-560.

[2] P. M. Chen, etc., "RAID: High-Performance, Reliable Secondary Storage", ACM Computing Surveys, Vol. 26, No. 2, June 1994.

[3] DECdirect Workgroup Solutions Catalog, Winter 1993.

[4] J. Gray, etc., "The 5 Minute Rule for Trading Memory for Disk Accesses And the 10 Byte Rule for Trading Memory for CPU Time", Proceedings of 1987 ACM SIGMOD Conference, San Francisco, CA, May 1987, pp. 395-398.

[5] J. Gray, A. Reuter, "Transaction Processing: Concepts and Techniques", Morgan Kaugmann Publishers, Inc. 1993.

[6] L. Gruenwald, M. H. Eich, "MMDB Reload Algorithm", Proceedings of ACM SIGMOD International Conference on Management of Data, May 1991, pp. 397-405.

[7] S. Gukal, E. Omiecinski, U. Ramachandran, "LU Logging - An Efficient Transaction Recovery Method", Technical Report GIT-CC-93, College of Computing, Georgia Institute of Technology, 1993.

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

[9] J. Huang, L. Gruenwald, "Logging Real-Time Main Memory Databases", Proceedings of International Computer Symposium, December 1994, pp. 1291-1296.

[10] J. Huang, L. Gruenwald, "Real-Time MMDB Reloading Techniques", Technical Report, School of Computer Science, University of Oklahoma, October 1995.

[11] X. Li, M. H. Eich, "Partition Checkpointing in Main Memory Databases", Technical Report, 93-CSE-23, Department of Computer Science and Engineering, South Methodist University, 1993.

[12] 22000 Series - SCSI Micropolis Disk Drive Information, 1993.

[13] A. Alen B. Pritsker, "Introduction of Simulation and SLAM II", John Wiley & Sons, Inc., New York, 1986.

[14] K. Ramamritham, "Real-Time Database", Invited Paper in International Journal of Distributed and Parallel Database, Vol. 1, No. 2, 1993, pp. 199-226.