CRASH RECOVERY FOR REAL-TIME MAIN MEMORY DATABASE SYSTEMS

Jing Huang Le Gruenwald

School of Computer Science
The University of Oklahoma
Norman, OK 73019
Email: gruenwal@mailhost.ecn.uoknor.edu

Keywords: Real-Time, Database, Recovery

ABSTRACT

In this paper we propose an update-frequency partition checkpoint scheme and a partition reload algorithm for real-time main memory databases (MMDB) which aim at not only reducing system recovery time, but also minimizing the number of timing constraints which are violated. With the update-frequency partition checkpoint scheme, an MMDB is divided into partitions based on data types (persistent vs. temporal) and update frequency, and each partition is checkpointed independently based on its update frequency. The partition reload algorithm allows the system to be brought up only when high access frequency partitions are reloaded into main memory and recovered. It takes transaction priority, reload prioritization and preemption into account during the reload process so that urgent transactions can be given immediate attention and have more chances to meet their deadlines. Our simulation results show that the proposed checkpoint technique outperforms the conventional fuzzy checkpoint approach. The partition reload scheme has a potential to provide a significant performance improvement over conventional reload.

1. INTRODUCTION

A real-time database system (RTDBS) is the one in which transactions not only maintain the consistency constraints of the database but also satisfy their timing constraints (e.g. deadlines). In addition to transaction deadlines, an RTDBS often processes both temporal data which lose their validity after a certain period of time, and persistent data which remain valid regardless of time [11]. The main goal of an RTDBS is to meet the timing constraints of transactions and data. Over the years, many research efforts have been made on developing efficient scheduling and concurrency control schemes for an RTDBS, but not much work has been done for recovery. As the cost of semiconductor memory decreases, it is possible for some real-time applications to keep the entire database in main memory (MM) so that a substantial performance improvement can be achieved. However, when a system failure occurs, as the database is not available for transactions, many transactions may miss deadlines and a large amount of temporal data may lose their validity before they can be used. The efficiency of a recovery technique therefore has a crucial impact on the performance of the system. In this paper we concentrate on two important components of a recovery mechanism: checkpointing and reloading.

Checkpointing is a process used to maintain on disks an up-to-date copy of the database and thereby provides a starting point for log recovery. When a system crash occurs, as checkpoints provide an almost up-to-date copy of the database, most data in the log are not needed at the time of recovery. The recovery process needs to process only the log information which is generated after the last checkpoint. Without an efficient checkpoint scheme, much recovery time will be consumed by log processing and consequently will result in many missing deadline transactions and invalid temporal data upon recovery. Reloading, which reloads a backup copy of the database from archive storage into main memory, takes place when a system failure occurs which causes the entire or partial contents of the MMDB to be lost. Efficient reloading is expected, especially in an MMDB environment, as transaction executions cannot proceed if required data are not memory-resident,

which if not handled properly may yield poor overall system performance. In this paper, we propose the update-frequency partition checkpoint (UFPC) scheme and the partition reload scheme for a real-time MMDB system. The objective of our approaches is not only to reduce the system restart time, but also to minimize the number of timing constrains which are violated. The idea of partition checkpoint was also proposed in ([7], [8]); however, not like ours, neither transaction deadlines nor temporal consistency requirements are considered in their schemes.

Simulation experiments are conducted to measure the performance of the proposed checkpoint and reload schemes. The remainder of this paper is organized as follows. We start in Section 2 with a description of media organization required by the UFPC scheme, then present the UFPC and reload algorithms in Sections 3 and 4, respectively. The simulation model will be introduced in Section 5. Performance experiments and results are discussed in Section 6. Lastly, Section 7 concludes the paper.

2. MEDIA ORGANIZATION REQUIRED BY THE UFPC SCHEME

As shown in Figure 1, an MMDB, which resides in the volatile Main Memory (MM), is divided into partitions, each of which contains one type of data only: persistent or temporal. Both persistent data and temporal data are partitioned based on their update frequencies. Pages which belong to one partition are not necessarily placed next to each other in MM, but are linked together so that they can be located easily during the checkpoint process.

Due to the volatility of semiconductor memory, the backup copy of the MMDB 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 [4], which copies the database in MM to AM periodically, is assumed in this work.

To facilitate database recovery, each partition has its own checkpoint bit map and local log buffer. The checkpoint bit map, which represents information about dirty pages during normal processing, is used to assist the implementation of fuzzy checkpoint. Log records generated during normal processing are grouped according to partitions and stored in the corresponding local log buffers. Based on our previous studies [5], 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 separately, gives the best performance in terms of log space, number of memory references and cost to perform REDO operations. This logging scheme is therefore assumed in this study. Log buffers are stored in a nonvolatile memory or stable memory (SM) and large enough to contain all updates of active transactions. When a log buffer is full, its contents will be flushed to a log disk. The global checkpoint record is used to record the location of the last complete checkpoint for each partition. By using this information, the starting point for post-crash log processing in each partition can be located easily. The recovery bit maps and recovery buffers are generated at the time of database reloading and used for recovery purposes.

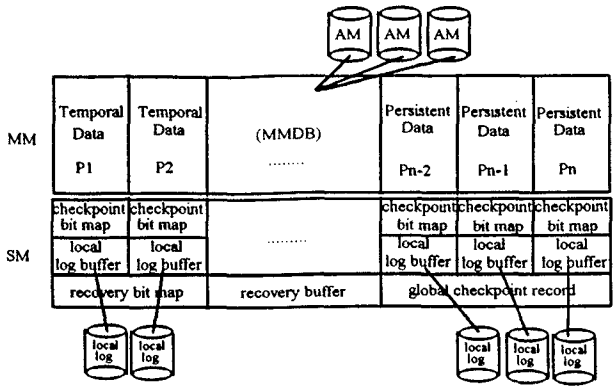

Figure 1. Media Organization Required by the UFPC Scheme

There exists a Database Processor (DP) and a Recovery Processor (RP) running in parallel in the system. The DP handles normal transaction execution while the RP manages transaction termination, logging, checkpointing and recovery from a system failure.

3. THE UFPC SCHEME

In the UFPC scheme, both persistent data and temporal data are partitioned based on update frequency. Partitions with a high update frequency (hot partitions) will be checkpointed more frequently than the partitions where updates seldom occur. The recovery procedure can then redo less information for hot partitions than would be required without partition checkpointing. The time needed to perform post-crash log processing is therefore reduced and the recovery process is hastened. This enables many transactions to meet their deadlines and a great amount of temporal data to be used before losing their validity.

Let UF_i and CF_i be partition i's update frequency which is the sum of update frequencies of all pages in partition i and checkpoint frequency, respectively. CF_i is determined based on UF_i in such a way that if $UF_i > UF_j$ then $CF_i > CF_j$; besides, CFi is assigned so that each partition can be checkpointed at least once in a limited period of time. In detail, the UFPC scheme consists of the following steps:

1. Partition persistent data and temporal data based on their update frequencies.
2. Calculate update frequency UF_i for each partition i.
3. Set checkpoint frequency CF_i of each partition i based on UF_i as follows: $CFi = \left\lceil \left(UFi \Big/ \sum_{i=1}^{Npart} UFi \right) * Npart \right\rceil$.
4. Find a partition i which has the highest checkpoint frequency among N_{part} partitions, that is

$CFi >= CFj, \quad j = 1, 2, \ldots, i-1, i+1, \ldots, N_{part}$

5. Invoke a local checkpoint on partition *i*.
6. Set CFi = CFi -1.
7. If there exists a partition *i* such that $CF_i > 0$, goto Step 4; Otherwise, goto Step 8.
8. Reset checkpoint frequencies CF_i's for all partitions i's to their original checkpoint frequencies and goto Step 4.

4. PARTITION RELOAD ALGORITHM

The key features of partition reload include 1) the system is brought up 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 chances to meet deadlines and 3) partitions accessed frequently are reloaded before other partitions so that the number of page faults can be reduced and transaction can be processed with fewer interruptions.

Before describing the partition reload algorithm, we first introduce the terminology that will be used in our discussion. *Transaction priority* is assigned based on transaction's execution priority. *Reload prioritization* indicates which data will be reloaded first and which data will be reloaded next. *Reload preemption* means that reload of some data is suspended and replaced by reload of some other data. *Reload threshold* is a variable which specifies the amount of data or the number of partitions that must be memory-resident before the system can be brought up.

To facilitate the reload process, the *disk striping* technique [12], which distributes data 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, following these cylinders are user data cylinders in which the backup copy of the MMDB resides. In order to reload each partition from AM into MM in the shortest amount of the time, pages belonging to the same partition should be placed contiguously in AM to reduce the disk arm movement overhead.

In detail, the partition reload algorithm consists of the following steps:

1. Reload system pages into MM on a cylinder basis (Performed by RP).
2. Reload the rest of the database based on access frequency. Partitions with higher access frequencies are brought into MM before partitions with lower access frequencies. Post-crash log processing for a partition is performed immediately after the partition is completely in MM. Reload of the next partition cannot start until the current reloaded partition is in MM and its consistent state is re-established. In this step, the reload of a partition (Performed by RP) and the reload of its corresponding log information (Performed by DP) can be performed in parallel.
3. Bring the system up when the reload threshold is reached.
4. Reload the rest of the database based on the following prioritization until the entire database is in MM (Performed by RP):

a) Demand reload priority (higher reload priority): Demand reload is based on transaction execution priority. When a page accessed by a transaction is not in MM, the execution of the transaction is suspended until the entire partition in which the demanded page resides is reloaded into MM and recovered. The RP is assigned to a transaction based on its execution priority, that is, the transaction with the highest execution priority will be placed in front of the waiting queue of RP.

b) Prefetch reload priority (lower reload priority): Prefetch reload the rest of the database based on access frequencies of partitions.

Demand reload priority is always higher than prefetch reload priority. The demand reload priority of a partition is the execution priority of the transaction which requests the partition. Since a partition is a recovery unit, if a page required by a transaction is currently being reloaded by prefetch reload, the transaction needs to wait until the current reload is finished. As prefetch reload has a lower priority, the waiting time could be indefinitely long which may finally cause the transaction to miss its deadline. To minimize this waiting time, we allow the prefetch reload process to inherit the priority of the requesting transaction so that the reload of the expected partition can be hastened. After the corresponding partition is recovered, prefetch reload will use its original, also the lowest, priority. In order to reestablish the consistent state of a partition, the log information that is stored in the corresponding local log buffer can be used. This is because in this algorithm, transactions are not allowed to access a partition before it is brought into MM and recovered, the contents of the log buffer remain unchanged.

5. THE SIMULATION MODEL

To measure the performance of the proposed checkpoint and partition reload schemes, we developed a simulation model of a real-time MMDB 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* is used for transaction execution priority assignment. 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 [2] and Micropolis 22000 disk drivers [9], since they accommodate high performance applications. The detailed explanation about parameter settings can be found in [6].

There are two types of transactions in the system: aperiodic and periodic. *Aperiodic* 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. If a normal transaction is found to read an invalid temporal data page, it 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. *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 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.

The simulation model is written in the SLAM II simulation language [10]. In each experiment at least 20,000 normal transactions are executed and 95% confidence intervals are obtained. The width of the confidence interval of each data point is less than 5% of the point estimate. The performance metric used is the percent of transactions missing deadlines, which is observed from the beginning of a simulation run until the end of the simulation. The conventional fuzzy checkpoint algorithm is selected to compare with the UFPC scheme. The reload algorithm called *conventional reload* in this study, which reloads the entire database into MM before bringing the system up, is chosen as a comparison.

6. SIMULATIONS RESULTS

Two sets of experiments are conducted to evaluate the behavior of the UFPC scheme and the partition reload algorithm. Figure 2 shows a comparison of the UFPC scheme with 10 partitions and the conventional checkpoint scheme under the various transaction arrival rates. As we can see, the performance improvement obtained by using the UFPC scheme is significant. Since each partition is checkpointed individually, the partition with a high number of updates can be flushed to the AM as frequently as possible, and finally leads to less recovery time than it would be required in conventional fuzzy checkpoint. The overall performance, in terms of transaction missing deadlines, is also improved due to fast restarts. The effects of write probability on the checkpoint schemes are demonstrated in Figure 3. The results obtained verify again that UFPC outperforms the conventional fuzzy checkpoint approach throughout the spectrum of write probabilities. In addition, this testing case indicates that the more partitions the database has, the better performance the system obtains. This is because the smaller a partition is, the less time is needed to checkpoint a partition, and the easier for the system to keep the most recent backup copy of hot partitions. However, UFPC does impose an overhead on normal system operations, and the more partitions the database has, the more overhead it incurs. When the number of partitions reaches a certain limit, the benefit obtained from further partitioning is not very significant.

Figure 4 shows the impact of the transaction arrival rate on the performance of the two reload algorithms. The reload threshold selected in this experiment is 50% of the database size. As we can see, when transaction arrival rate is high, the system is overloaded by a huge number of transactions striving to access a limited number of resources. Consequently, many transactions miss their deadlines. The results also indicate that the partition reload algorithm consistently performs better than the conventional reload approach. This is because that partition reload allows transaction execution to be resumed earlier than conventional reload does. In partition reload, the system is brought up only when some frequently updated partitions are reloaded into MM and recovered, while the other partitions are brought into MM on demand or in background. Since transaction processing and database recovery can be performed in parallel, recovery performance is improved.

We also examined how the different number of partitions affects the performance of the partition reload algorithms. The results obtained show that the overall performance of partition reload is improved as the number of partitions is increased. However, this algorithm does not consistently yield better results than the conventional reload approach. When the number of partitions is a small number (smaller than 6), this algorithm gives the worst results among the three alternatives. Its performance starts getting better than that incurred in conventional reload when the number of partitions is more than 8. This can be explained as follows. In the partition reload scheme, when a page fault occurs, the execution of the requesting transaction must be suspended until the entire partition in which the requested page resides is reloaded into MM and recovered. The fewer partitions the database has, the larger a partition is, and accordingly the longer time the requesting transaction needs to wait. Furthermore, since the database reloading and transaction processing will interfere with each other in this algorithm, which slows down the reload process and

subsequently leads to the worse overall performance compared with that incurred in the conventional reload scheme. Therefore, in order to take advantage of partition reload, the number of partitions must be selected carefully and should be a large number.

7. CONCLUSIONS

In this paper, we proposed a update-frequency partition checkpoint (UFPC) scheme and a partition reload algorithm for a real-time MMDB. The aim of our approaches is not only to reduce the crash recovery time but also to enable many transactions and data to meet their timing constraints. The simulation results obtained confirm the superiority of our proposed approaches. In particular, they indicated that the UFPC scheme consistently outperforms the conventional fuzzy checkpoint approach. The more partitions the system has, the better performance is obtained. However, at a certain point, the benefit obtained from further partitioning is lessened as the number of partitions is continuously increased. The partition reload algorithm has a potential to offer better recovery performance than conventional reload does. In general, the more partitions the database is divided, the better performance the partition reload scheme can provide. Furthermore, the number of partitions is a crucial factor that affects the behavior of partition reload. If this number is not selected carefully, too many page faults will be incurred, which subsequently will outweigh the gain in system unavailability, and thus will lead to a worse overall system performance.

REFERENCES

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

[2] *DECdirect Workgroup Solutions Catalog*, 1993.

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

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

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

[6] Huang, J., 1995, "Recovery Techniques in Real-Time Main Memory Databases", Ph.D. Dissertation, School of Computer Science, The University of Oklahoma.

[7] Jagadish, H. V., Silberschatz, A., Sudarshan, S., 1993 "Recovering From Main Memory Lapse", *Proceedings of the 19th VLDB Conference*, pp. 391-404.

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

[9] *22000 Series - SCSI Micropolis Disk Drive Information*, 1993.

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

[11] Ramamritham, K., 1993 "Real-Time Database", *Journal of Distributed and Parallel Database*, Vol. 1, No. 2, pp. 199-226.

[12] Salem, K., Garcia-Monlina, H., 1986, "Disk Striping", *Proceedings of the 2nd International Conference on Data Engineering*, February.

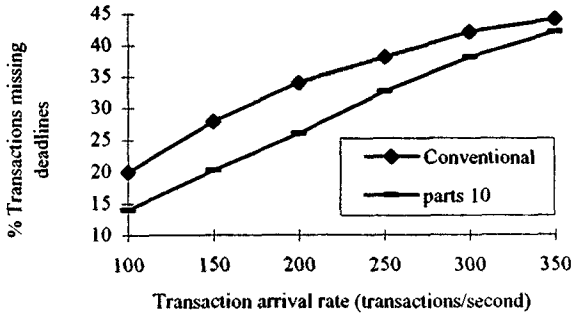

Figure 2. Transaction Arrival Rate vs. Percent of Transactions Missing Deadlines

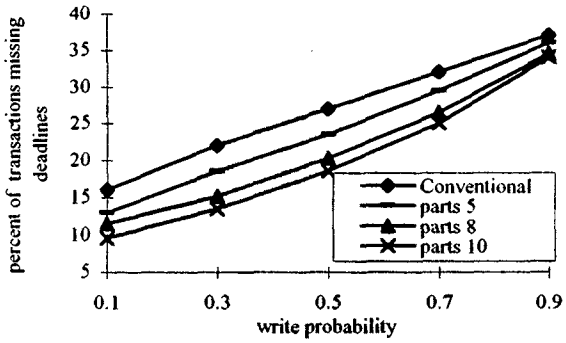

Figure 3. Write Probability vs. Percent of Transactions Missing Deadlines

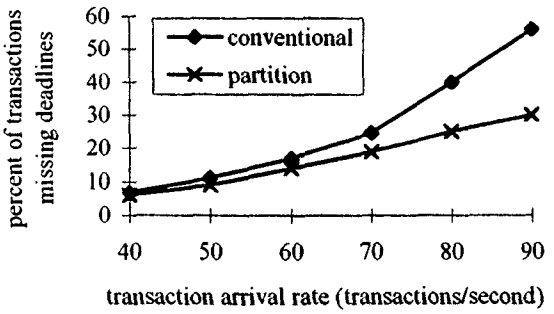

Figure 4. Transaction Arrival Rate vs. Percent of Transactions Missing Deadlines (Reload Algorithm)