Correspondence

Effects of Update Techniques on Main Memory Database System Performance

Le Gruenwald, *Member*, *IEEE Computer Society*,
YuWei Chen, and Jing Huang

Abstract—Update technique is an important issue related to database recovery. In a main memory database environment, transaction execution can be processed without any I/O, and all I/O operations involved are for recovery purposes. The efficiency of update techniques therefore has an important impact on the performance of main memory database systems. In this paper, we compared the techniques of immediate and deferred update based on a database machine, MARS. The simulation results showed that immediate update outperforms deferred update unless system failure is a frequent occurrence.

Index Terms—Main memory databases, update, failure recovery, transaction processing, simulation.

1 Introduction

TWO commonly known update techniques in a database system are immediate update and deferred update. The immediate update approach allows database modifications made by a transaction to be output to the database while the transaction is still in an active state. Deferred update keeps changes in a separate area until a successful completion of the transaction is assured, at which time the modifications are applied to the database. It is generally agreed that the deferred update approach offers a poor performance in disk-resident databases (DRDBs) and immediate update is commonly used [6]. However, not much research has been done in main memory database systems (MMDBs) where the entire or a major portion of the database is memory-resident. A major difference between an MMDB and a DRDB is that transactions in an MMDB commit their results in main memory while transactions in a DRDB commit on disk. Thus, in an MMDB, I/O operations incurred in performing update policies are reduced tremendously, especially for deferred update, compared with that in a DRDB system. Update polices may behave differently in MMDBs. It is therefore necessary for us to re-examine these two policies in the new environment.

In this paper we conduct a simulation study based on the MMDB system, MARS [2] to evaluate the performance of the two update approaches in an MMDB in the presence of a system crash. The outline of this paper is as follows. Section 2 presents the system assumptions and the recovery algorithm used in this study. Section 3 describes our simulation model and parameter settings. Section 4 analyzes the simulation results and, finally, Section 5 summarizes the paper.

• *The authors are with the School of Computer Science, University of Oklahoma, 200 Felgar St., Rm. 114 EL Norman, OK 73019. E-mail: gruenwal@cs.ou.edu.*

Manuscript received 20 July 1995; revised 4 Mar. 1996.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 104328.

2 System Assumptions and Recovery Algorithm

MARS (MAin memory Recoverable database with Stable log) [2] is the main memory database system that we have simulated. It assumes that the entire database resides in a volatile main memory (MM), while its backup copy is kept in an archive memory (AM) residing on secondary storage. Transaction processing is manipulated by the Database Processor (DP) and recovery activities including logging, checkpoint, and recovery are handled by the Recovery Processor (RP). The log buffer is stored in the nonvolatile or stable memory and is large enough to contain all updates of active transactions.

With immediate update, modified pages may be propagated to the primary database at any time. Hence, in order to make failure recovery possible, Before Images (BFIMs) are saved in a log file before they are overwritten by After Images (AFIMs). The commit processing of transactions is trivial. However, to abort a transaction, the RP needs to rollback all of its actions. Both UNDO and REDO operations are required at the time of a system crash. With deferred update, modified data is kept in the log until a successful completion of the transaction performing the updates is assured. Since no dirty pages are propagated to disks, only AFIMs need to be logged for REDO purposes.

Fuzzy checkpoint [5], which does not require the system to be quiescent during the checkpoint process, is assumed in this work. In order to reload the database into MM after a crash, the frequency reload algorithm is used. This algorithm is selected as it yields the best system performance in terms of transaction response time and system throughput compared with other reload algorithms proposed in [3]. In frequency reload, data is reloaded according to some priority order and the system resumes its execution when a certain amount of the database is memory-resident. It takes reload prioritization, preemption, and access frequency into account. The detailed description of the algorithm can be found in [4].

3 Description of the Simulation Model

The simulation model used to evaluate the performance of the two update techniques is based on those constructed in [3], which are written in the simulation language SLAM II [7]. In order to investigate the behavior of the immediate update and deferred update schemes in both cases, with and without a system failure, at least 20,000 transactions are executed and the 95 percent confidence intervals are obtained. The width of the confidence interval of each data point is less than 5 percent of the point estimate. The performance metric measured is average transaction response time, which includes times needed for transaction processing, resource waiting, logging, commit, and UNDO (for immediate update only).

In our simulation, transaction arrival rates are exponentially distributed. The size of a transaction is determined by the number of operations it executes, which is distributed uniformly between 5 and 30. Transaction concurrency control is accomplished through conservative two-phase locking [1]. Since each transaction performs operations on pages, a page-level lock granularity is chosen.

All the simulation parameters, which are mostly adopted from existing literature, are listed in Table 1 and Table 2. The default values of these parameters, such as MM_ACCESS, CPU_POWER, disk parameters, are selected based on the DEC 3000 Model 400/400S AXP Alpha machine and Micropolis 22000 disk's drives,

TABLE 1
DYNAMIC PARAMETERS

Parameter	Default Value	Range	Parameter	Default Value	Range
Number of AM disks	2	1 to 5	Database Size	1,800	600 to 5,400 pages
Write Probability	20%	0% to 100%	Read Probability	80%	100—Write Probability
Multiprogramming Level	10	1 to 20	Transaction Abort Rate	3%	0% to 40%

TABLE 2
STATIC PARAMETERS

Parameter	Meaning	Value	Parameter	Meaning	Value
SM_ACCESS	Access an SM word	0.000198 ms	SM_SEAR	SM address translation	0.5* MM_ACCESS
ALLOC_TM	Allocate a MM page	0.005 ms	MM_SEAR	MM address translation	3* MM_ACCESS
AMREQ_TM	Request an I/O from AM	0.00143 ms	PRETRAM	Preprocess a transaction	0.0072 ms
RELEASE_TM	Release an MM page	0.005 ms	PREOP	Preprocess an operation	0.000007 ms
BMAP_TM	Read until 1 in bit map	0.02957 ms	ET_TM	End transaction	0.0054 ms
MM_ACCESS	Access an MM word	0.00018 ms	INTIO_TM	Initiate log I/O	0.0014 ms
REC_SZ (deferred)	SM or log record	12 bytes	LOGPG_SZ	Log page size	2,000 bytes
REC_SZ (immediate)	Log record	16 bytes	WORD_SZ	Bytes per word	4 bytes
BLOCKS_CYL	Blocks per Cylinder	30	LOGIO_TM	Write a log page	5.624 ms
TRACKS_CYL	Tracks per Cylinder	15	CYLS_AM	Cylinders per Disk	30
TRANSFER	Transfer time	7.46 ms	SEEK	Average seek time	10.0 ms
INDN_TM	Initial down time	5.0 ms	LATENCY	Average latency	5.56 ms
LOCK/UNLK_TM	Get/release one lock	0.0007 ms	HASH_TM	Hash AM directory	0.0007 ms
SIGNAL_TM	DP signals RP for a page fault	0.0007 ms	CPU_POWER	CPU_POWER	140 MIPS

since they accommodate high performance applications. The detailed explanations of the parameters can be found in [4].

4 SIMULATION RESULTS AND DISCUSSIONS

In this section, we present the simulation results which best illustrates the system performance. Fig. 1 illustrates how transaction arrival rate affects transaction response time before a system crash. It is obvious that when transaction arrival rate is high, the system is overloaded by a huge number of transactions striving to access a limited number of resources, transaction response time increases accordingly. In Fig. 1, the simulation result shows that immediate update outperforms deferred update when there is no system crash. This is because in deferred update, a transaction commit involves accessing the SM and creating log records in the log buffer, while in immediate update, after finishing all operations successfully, a transaction can commit immediately. A higher workload in a transaction commit accounts for a higher transaction response time in deferred update during normal transaction processing.

In order to determine how much overhead database recovery will impose on normal transaction processing, we observed average transaction response time during different time intervals after a crash. This is because of the fact that the recovery process has more impact on transactions that arrive right after a system failure than those that come later and the effect of recovery overhead is

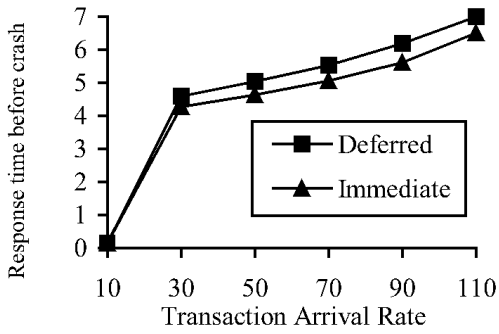

Fig. 1. Transaction arrival rate vs. transaction response time before crash.

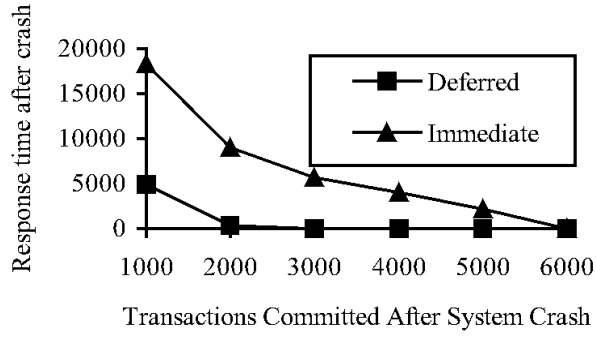

Fig. 2. Time interval after system crash vs. transaction response time.

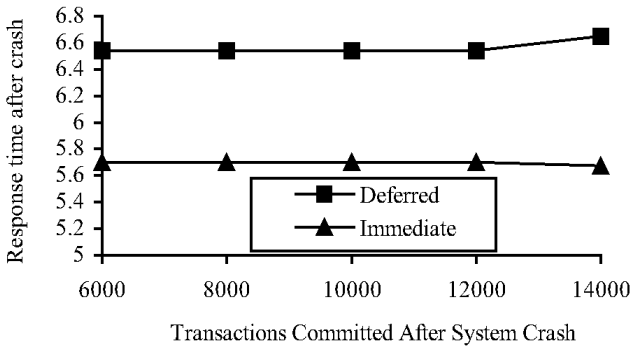

Fig. 3. Time interval after system crash vs. transaction response time.

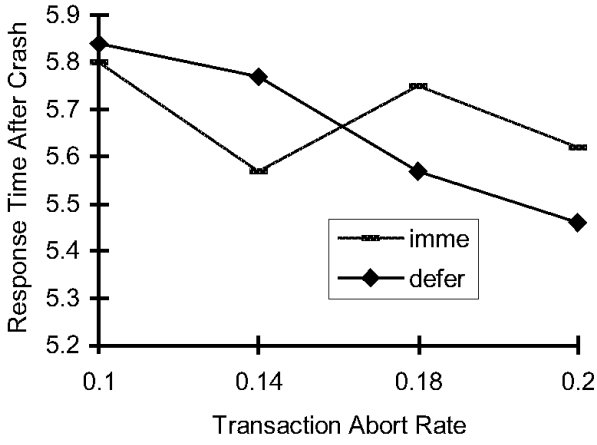

Fig. 4. Transaction abort rate vs. transaction response time.

diminished as the time passes. The results obtained show that deferred update performs better than immediate update only during a very short time interval after system recovery (Fig. 2); however, when the system resumes its normal and stable state, immediate update behaves better than deferred update (Fig. 3).

We also conducted a testing case to examine how transaction abort rate affects the performance of the two update policies. The simulation results indicated that when the transaction abort rate is lower than 15 percent, immediate update yields better transaction response time than deferred update does (Fig. 4).

5 Conclusions

We examined the performance of the two update algorithms, immediate update and deferred update, in a main memory database system MARS. Our results indicated that the immediate update policy provides a better average response time for normal transaction processing when the transaction abort rate is lower than 15 percent. In the existence of a system crash, due to less log information that needs to be processed during recovery time and the use of a stable memory, deferred update enables the system to resume earlier than immediate update does, which finally results in a faster average response time. However, when the system is back to a stable state after a system crash, immediate update again outperforms deferred update. As transaction abort rate is usually lower than 5 percent in conventional database systems, the choice of an update scheme depends only on system failure rate. We can therefore draw the following conclusion: In MMDBs, unless system failure is a frequent occurrence, immediate update should be used.

Acknowledgments

This material is based, in part, on work supported by the National Science Foundation under Grant No. IRI-9201596.

References

[1] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

[2] M.H. Eich, "MARS: The Design of a Main Memory Database Machine," *Proc. Int'l Workshop Database Machines*, Oct. 1987.

[3] L. Gruenwald and M.H. Eich, "MMDB Reloading Algorithms," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 397-405, June 1991.

[4] L. Gruenwald et al., "Evaluation of Reloading and Paging in Main Memory Database Systems," *J. Brazilian Computer Soc.*, special issue on database systems, vol. 2, no. 3, pp. 24-35., Apr. 1996.

[5] R.B. Hagmann, "A Crash Recovery Scheme for a Memory-Resident Database System," *IEEE Trans. Computers*, vol. 35, no. 9, pp. 839-343, Sept. 1986.

[6] V. Kumar and A. Burger, "Performance Measurement of Main Memory Database Recovery Algorithms Based on Update-in-Place and Shadow Approaches," *IEEE Trans. Knowledge and Data Eng.*, vol. 4, no. 6, pp. 567-571, Dec. 1992.

[7] A. Alan and B. Pritsker, *Introduction to Simulation and SLAM II*, System Publishing, 1986.