Simulation of Main Memory Database Parallel Recovery

QIN Xiongpai, CAO Wei, WANG Shan
Key Laboratory of Data Engineering and Knowledge Engineering (School of Information, Renmin University of China), MOE, Beijing 100872, P.R.China
qxp1990@sina.com, {caowei,swang}@ruc.edu.cn

Keywords: Large-scale main memory database, update intensive, parallel recovery, discrete event simulation

Abstract

The paper describes the details of using J-SIM in main memory database parallel recovery simulation. In update intensive main memory database systems, I/O is still the dominant performance bottleneck. A proposal of parallel recovery scheme for large-scale update intensive main memory database systems is presented. Simulation provides a faster way of evaluating the new idea compared to actual system implementation. J-SIM is an open source discrete time simulation software package. The simulation implementation using J-SIM is elaborated in terms of resource modeling, transaction processing system modeling and workload modeling. Finally, with simulation results analyzed, the effectiveness of the parallel recovery scheme is verified and the feasibility of J-SIM's application in main memory database system simulation is demonstrated.

1. INTRODUCTION

Main memory database systems are widely used in update intensive applications [1] such as telecom, electric power industry, financial industry, and defense. Due to high throughput of main memory database systems, the volume of the log records generated is huge. A poor logging scheme will severely affect transaction processing performance.

Memory capacity gets larger and larger, and the price of memory is decreasing. Cheaper and larger memory makes large-scale main memory database systems a reality. Main memory database systems of terabyte size have been tested, to recover such a large-scale main memory database system, total recovery time will be hours [2]. As part of the benchmark test in [2], engineers restored a database of 1.17 terabytes in 4.76 hours. Not only should the total recovery time be cut down, but also new transactions should be handled during recovery. It is not tolerable in high performance applications for new transactions to have to be blocked until the recovery process finishes.

The IT industry has witnessed huge progress of hardware in recent years, including multi-core processors [3], non-volatile memory of large capacity [4], and solid-state disks [5]. New hardware brings opportunity to main memory database research. A parallel recovery scheme for update intensive main memory database systems, which is adaptive to new hardware, is presented. The scheme makes use of the nice parallel properties of differential logging. Log records are grouped by data partition IDs and dispatched to multiple log disks to improve logging efficiency and speedup transaction committing. During recovery, data partitions are recovered in parallel according to loading priorities, total recovery time is minimized, blocked transactions could resume running after needed data partitions are recovered, and the scheme provides system availability during recovery.

Database system implementation runs into millions of lines of code. Incorporating the idea of parallel recovery into existing DBMS may take a few man-years. Simulation provides a faster way of evaluating the proposed idea. It is easy to vary the resources (CPU, memory, disks and so on) and workload parameters to find out the impact in a simulation.

2. THE PARALLEL RECOVERY SCHEME

Figure 1 shows the usual main memory database systems architecture. The primary database resides in the physical memory. Log records generated during transaction processing are temporarily saved to some form of non-volatile memory to speedup transaction committing. The log records will be written to log disks asynchronously later. With the inherited parallel properties of differential logging, the log records can be distributed to multiple log disks. The whole database is organized with a *Segment-Partition-Page* scheme. Data partition is the basic unit for checkpointing. Data partitions can be distributed to different data disks.

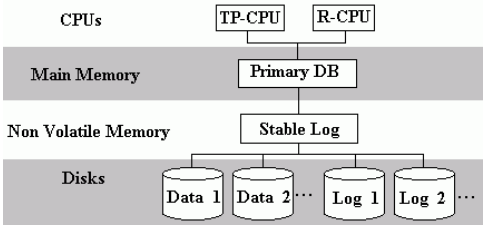


Figure 1. System Architecture

A database is comprised of several segments. Each segment contains data and index of one relation in the system. The segment is a collection of data partitions. Each data partition contains fix number of data pages. Every record (tuple) has a unique RID, which takes the form of

<Segment Number, Partition Number, Page Number, Slot Number>, and the record slot in the page contains the corresponding record's offset and size.

We focus on how to build the simulation system for parallel recovery experiments here. The parallel recovery scheme is introduced in brief. Readers who are interested in the details of the scheme could refer to reference [6].

2.1. Parallel Logging

Parallel logging is based on differential log. If an update transaction has changed a data item's value from B (Before image) into A (After image), then the differential log is defined as B $\oplus$ A, $\oplus$ denotes bit-wise XOR.

XOR has several nice properties. (1) Existence: V $\oplus$ 0 =V. XOR could be used to set the value of a data item which is initialized with 0 bits. (2) Commutative: V1 $\oplus$ V2= V2 $\oplus$ V1. (3) Associative: (V1 $\oplus$ V2) $\oplus$ V3= V1 $\oplus$ (V2 $\oplus$ V3).

Owing to the commutative and associative properties of XOR, applying differential logs to data items could be in an arbitrary order, which implies that the log records can be partitioned at will and dispatched to multiple log disks to improve efficiency of logging. During recovery, the log records could be read from log disks in parallel and applied to data without the need of reordering, which would speedup recovery. Parallel logging and parallel recovery are especially beneficial for update intensive main memory database systems in which disk access is still the dominant bottleneck of system performance.

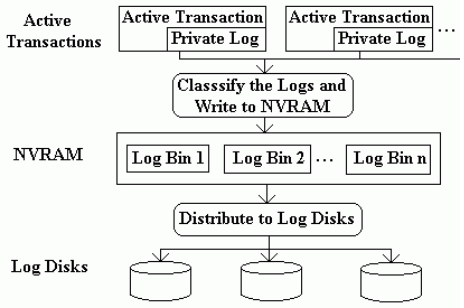


Figure 2. Parallel Logging

Log records generated during transaction processing are maintained privately by respective transactions to avoid contention on the *Global Log Tail* before transaction commit. When a transaction enters the commit phase, the log records of the transaction are saved to non-volatile memory to accelerate committing. In case of transaction abort or rollback, the log records are simply discarded. Log records are partitioned according to data partition IDs and saved in corresponding log bins, when the log bin for some data partition is full, the log records are written to log disks asynchronously. Parallel log writing improves logging efficiency.

2.2. Partition Based Consistent Checkpointing

(1) Prepare for Consistent Checkpointing: A dynamic multi version two-phase lock protocol is used to schedule transactions. Multi versioning is the precondition of partition based consistent checkpointing. The protocol is based on transient version two-phase lock protocols for disk resident database systems [7]. A major enhancement is: Data partition is selected to be the only locking granularity. Such a locking granularity ensures high system concurrency as well as little locking overhead. A two-level version management scheme is employed to avoid version accumulation and accelerate version reuse. A dynamic version sharing technique, which is traditionally used to improve system concurrency, is used to further reduce space overhead of multi versioning. Under the control of the protocol, every read only transaction is assigned a startup timestamp. It reads the latest committed version of data items before the startup timestamp and return. Read only transactions do not need to obtain locks on data. Tuple level copy-on-update is used to support update transactions. When update transactions try to modify some data item, data is copy on tuple level. Two phase locking guarantees the serialization execution of update transactions.

(2) None Blocking Consistent Checkpointing: Data partition checkpointing is implemented as a read only transaction; it reads latest committed data of the partition and flushes it to disk to replace the stale data. A Ping-Pong scheme is used to maintain checkpoint files of data partitions. Checkpointing frequencies of data partitions are computed from update frequencies tracked during normal transaction processing. Data partitions that are updated frequently will get checkpointed more often whereas data partitions that are updated less frequently will get checkpointed much less, thus the volume of log records, which is needed to be processed during recovery, is reduced. To recover a data partition to the recent consistent state after system failures, the log records generated after the checkpoint timestamp should be applied to the data partition loaded from the checkpoint file. The log records should be applied once and only once, because of none-idempotence property of differential log. The [*Begin Checkpoint*] log record plays the role of redo point in the log files.

Since read only transactions do not incur lock conflicts with update transactions, checkpointing interferes little with normal transaction processing and high system throughput is expected. In addition, only committed data is flushed to checkpoint files, the checkpoint files are transaction consistent. The scheme shares the advantages of *Redo-Only Logging*, i.e. only redo logs of committed transactions need to be persistent during recovery. One pass of log scanning is enough. Recovery becomes simpler compared to fuzzy checkpointing, which needs two passes of log scanning (Undo Pass & Redo Pass).

2.3. Parallel Restart

Data partitions are loaded and restored according to computed loading priorities. The loading priority for every data partition is computed using the following equation (update frequency information is maintained in non-volatile memory):

$$Priority(j) = \alpha \frac{UF(j)}{\sum_{j=1}^{PART^{\#}} UF(j)} + (1-\alpha) \sum_{i=1}^{TX^{\#}} a_{ij} \frac{WT(i)}{\sum_{i=1}^{TX^{\#}} WT(i)} \quad (1)$$

(α =0.3, $PART^{\#}$: Partition Number; UF: Update Frequency; $TX^{\#}$: Active Transactions Number; WT: Waiting Time of a Transaction; a_{ij}: If T(i) is waiting for P(j) to be recovered, a_{ij}=1, else a_{ij}=0.)

The system starts some thread groups to recover data partitions having top-N loading priorities. After some data partition has been restored in memory, the thread group is reassigned the recovery task of a new data partition having the highest loading priority. It is worthy to mention is that loading priorities for the remaining data partitions should be recomputed because the waiting time has changed.

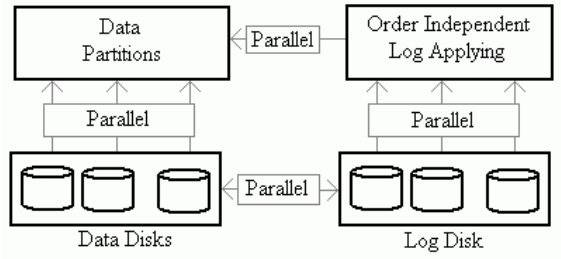


Figure 3. Parallel Restart

A thread group is responsible for the recovery of a data partition. Respective threads perform the tasks of checkpoint file loading, log file loading and log applying. Three types of parallelism are exploited to boost the recovery. (1) Different data partitions are loaded and recovered in parallel by respective thread groups. (2) The log records of one data partition can be loaded from multiple disks in parallel, and applied to the data partition in time, i.e., there is no need to reorder the log records according to the serialization order in memory. (3) The memory image of a data partition is initialized to all zero bits. After loading the checkpoint file, it is applied to the memory image of the data partition using the XOR operation. The log records are also applied to the data partition using XOR. Owing to the nice associative and commutative properties of XOR, loading & applying the checkpoint file, and loading & applying log records can run in parallel.

Once critical meta-data is restored, the system admits new transactions and gets them executed. If the data partitions needed by executing transactions have not been restored in main memory yet, the transaction manager blocks the transaction until the data partition is restored. The restart algorithm provides system availability during recovery. New transactions can be handled as soon as possible without the need to wait for a long time. Parallel recovery also cuts down transaction response times.

3. SIMULATION SYSTEM BUILDING

3.1. Introduction to J-SIM

J-SIM [8], an open source simulation package, is evolved from C-SIM. J-SIM provides basic classes for simulation, process and queue.

(1) Infrastructure of J-SIM: JSIMSimulation is the most important one of J-SIM's fundamental classes. A JSIMSimulation instance represents the simulation model where various numbers of processes and queues can be inserted. A calendar is owned by every simulation object, where events created by the simulation's processes are inserted.

JSIMProcess class is a template class for a user process. A method called life() is introduced in JSIMProcess, which contains the code representing the behavior of a process. A simulation can contain a number of independent active processes. Every process has it own pre-programmed life() method, which can be divided into parts. One part of a process's life() is executed at one exact point of simulation time, which does not change during the execution. All parts of all processes' life() are merged together and scheduled according to the value of their simulation time.

The execution of the simulation is divided into steps. One step corresponds to the execution of one selected process's part, the execution is under fully control of the currently executed process, and no other process can interrupt or postpone the execution. All processes share the calendar where events are stored. An event is an object holding information about a process's life part. The information contains the process's ID and the value of simulation time at which the life part is scheduled. In order to divide process life() into parts, the process uses reactivation routines (passivate(), hold(), activate()) to establish reactivation points in the code of their lives. The JSIMSimulation class provides a step() method to execute the simulation in a step-by-step manner. During one simulation step, exactly one event is interpreted and destroyed afterward.

The JSIMHead class represents the head of a queue where objects to be processed can be inserted. JSIMHead's useful functions include: empty(), cardinal(), first(), last(), and clear() which have been seen in Simula [9]. JSIMHead also provides getLW() and getTW(), returning the mean queue length *Lw* and the mean waiting time in the queue *Tw* respectively, which provide data for statistics.

JSIM also provides an extensive subsidiary class to facilitate simulation system building.

(2) Simulation Time: All processes within the same simulation share the same time. In main memory database systems, transactions finish in milliseconds, and instructions is executed in less than a millisecond. We map a time unit of the simulation to a microsecond (1/1000 millisecond).

(3) Modeling Strategy: We use the Active Station and Passive Customer (*ASPC*) strategy to model the main memory database system. Transaction is implemented as passive customers, they are enqueued into a schedule queue, waiting for service there. A service station comprises two parts: server and related queue. When a server (active process) finishes serving a transaction, it fetches the next transaction from its input queue. All servers are linked together to make a complex simulation system. After all operations have been executed, a transaction could commit and leave the simulation system.

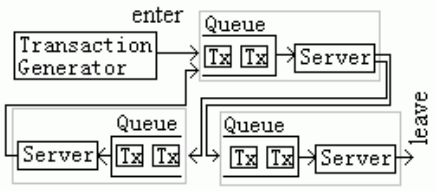


Figure 4. Active Station and Passive Customer

The main memory database simulation system is depicted in figure 5. The system includes fives modules, resource management (RM), transaction processing (TP), checkpointing processing (CP), log processing (LP), and recovery processing (RP).

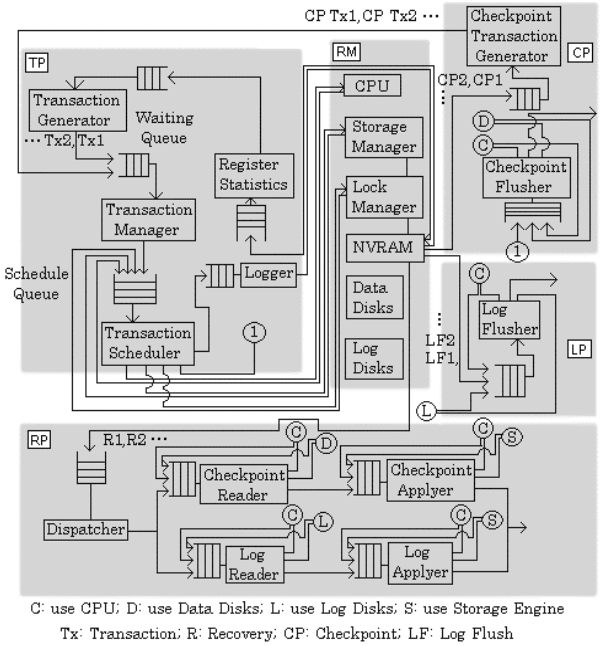


Figure 5. Simulation System Model

3.2. Resource Modeling

The resources of a database system include CPU, Memory (modeled as a storage engine), non-volatile memory and disks.

(1) CPU Modeling: CPU Modeling is depicted in figure 6, which can simulate multi-core CPUs. Transactions having instruction execution demands are inserted into the common queue, and then dispatched to the idlest CPU by the Scheduler. The Scheduler could also employ a different policy, for example, normal transaction processing is directed to the Transaction CPU, or recovery related work is directed to the Recovery CPU. The Scheduler and CPUs are implemented as active processes (JSIMProcess subclass).

(2) Disk Modeling: Several parameters including seek time, access latency, and sustainable transfer rate are used to model a disk. The simulation system contains a number of data disks and log disks, which act as storage for checkpoint files and log files respectively. IO requests are en-queued into a common queue and dispatched to the corresponding disk's queue.

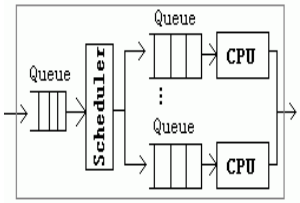


Figure 6. CPU Modeling

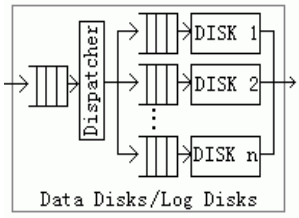


Figure 7. Disk Modeling

(3) Storage Engine Modeling (figure 8): Every data partition has an attribute named *inMem*, which is used for synchronization in recovery. When the data partition has been recovered into memory, the *inMem* attribute takes the value of YES. Transactions then can access the data partition. When the value of *inMem* attribute is NO, which indicates that the data partition has not been recovered from disk, transactions that try to access the partition have to wait in some data partition's Recovery Waiting Queue. A recovery task for the partition is generated and handed over to the Recovery Manager for further processing. When the data partition has been recovered, it is labeled as accessible by setting the value of *inMem* to YES. Some Access Handler is re-activated and waiting transactions are further processed.

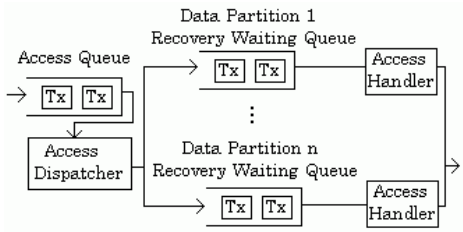


Figure 8. Storage Engine Modeling

(4) Non-volatile memory (NVRAM) Modeling: In modeling the NVRAM, the access latency is captured. We do not set a capacity limitation for the NVRAM.

(5) Lock Manager Modeling: For every data partition, the current lock holder as well as transactions that are waiting to lock it should be maintained. The flow of locking is that, if the current lock holder of the data partition is null, then the requesting transaction is de-queued from the Lock Queue. The requested lock is granted to it and current lock holder of the partition is set accordingly. When the current lock holder of the partition is not null, the requesting transaction is dispatched to the Lock Waiting Queue of the partition. When a lock is released, the Lock Waiting Queue is checked to see if some transaction is waiting for locking. If so, the next transaction is de-queued from the head of the queue and granted the lock. Deadlock detection is also implemented. Lock Dispatcher and Lock Handler are implemented as subclasses of JSIMProcess.

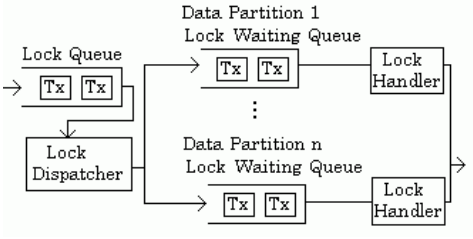


Figure 9. Lock Manager

Although the diagram of the Lock Manager and that of the Storage Engine follow the same pattern, their internal processing logics are different.

Resource contention could be simulated by varying the resource number. From example, we can reduce the number of CPU cores to imitate a shortage of computing power, or reduce disk number / disk sustainable transfer rate to imitate a shortage of IO capability.

Table 1. Parameters for Resource Modeling

Parameter	Meaning	Value
CPU#	Number of CPUs (cores)	2
CPU Speed	MIPS of a CPU	20K
NVRAM access time	Access latency of NVRAM	226 μ s(512B)
NVRAM Capacity	Capacity of NVRAM	Unlimited
Disk access latency	Disk access latency	4ms
Disk seek latency	Disk seek latency	8ms
Disk bandwidth	Disk sustainable R/W bandwidth	36MB/s
Data Disks#	Number of data disk	1, 2, 4, 6, 8
Log Disks#	Number of log disk	1, 2, 4, 6, 8
DB Size	Size of the database	131072 pages (2GB)
Partition Size	Size of a data partition	32pages
Page Size	Page size in bytes	16KB
Tuple Size	Tuple size in bytes	256B
Version Chasing	Cost for finding a tuple version	100 instructions

3.3. Transaction System Modeling

(1) Transaction Processing Modeling: Firstly, a number of transactions are generated by the Transaction Generator according to the *Update Transaction Percentage* parameter, and handed over to the Transaction Manager. The Transaction manager is responsible for scheduling the transactions to run concurrently using the dynamic multi version two-phase lock protocol. Locking requests are handled by the Lock Manager. When a transaction needs to use some resource (CPU, IO, Lock, Page Access), the transaction is put into related queues (CPU queue, IO queue, Lock Manager internal queue, Storage Engine internal queue). When some operation of a transaction is done, the Transaction Manager checks it to see if there exist any other operations to be done. If so, the transaction is put in the schedule queue for further processing, else it is removed from the system and related statistics information is recorded. When a transaction finishes, a new transaction will be generated to maintain the multi-programming level. The Transaction Manager and the Scheduler are implemented as subclasses of JSIMProcess.

(2) Log Processing Modeling: Parallel logging is implemented. Log records in NVRAM are classified according to data partition IDs. When the log bin for some data partition is full, the log records are flushed to log disks asynchronously by the background Log Flusher. The Log Flusher is implemented through inheriting from the JSIMProcess class.

(3) Checkpointing Modeling: Checkpointing transactions are also scheduled by the Transaction Manager. A dedicated Recovery CPU can be used to serve checkpointing tasks. A checkpoint transaction reads committed data of every page in the data partition and prepares the data in a buffer. Then an IO request is en-queued in the IO queue of data disks, which is handled by an active disk simulator later. The buffer is flushed to disk to renew the checkpoint file.

(4) Recovery Modeling: During recovery, new transactions are en-queued in the access queue of storage engine. These transactions have to wait to be accessed until data partitions are reloaded and restored to a consistent state. Then they are awoken to resume as depicted in figure 10.

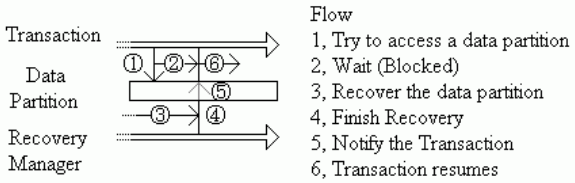


Figure 10. Recovery Modeling

3.4. Workload Modeling

Workloads of update intensive applications mentioned in section one, share common characteristics. There exist a large number of update transactions in the workload. Update transactions are usually small, but the time constraint is very stringent, ranging from several milliseconds to tens of milliseconds. In the workload, there exist two types of transactions, read only transactions (queries) and update transactions. The parameter *Update Transaction Percentage* is used to control the workload mix. We adopted a R20U80 mix in experiments, which denotes a

mix of 20% queries and 80% updates. The number of tuples touched by a transaction is controlled by the parameter *Transaction Size*, which follows a uniform distribution [5, 10]. The *Access Pattern* parameter is used to find out system performance behavior under the condition of data contention. An *Access Pattern* of 80:20 means that 80% of data accesses concentrate on some portion (20%) of the whole database.

(1) Transaction Modeling: A transaction is composed of operations such as begin transaction, commit transaction, abort transaction, lock/unlock data partitions, latch/unlatch data pages, and select/update tuples. Every operation executes some number of instructions and takes some time to finish, which is dependant on the power of the CPU.

(2) The Transaction Generator: A Transaction Generator drives the whole simulation. Two strategies could be used to generate transactions.

1) Think Mode: Think mode simulates real life user requests. A transaction is generated after a period of user think time. A number of transaction generator threads are used to simulate multiple user scenarios.

2) Pressing Mode: The mode tries to put a saturating workload on the target system. The number of concurrent transactions is specified by the *MPL (Multi Programming Level)* parameter. At the very beginning of the simulation, the Transaction Generator generates a number of transactions to fill the schedule queue. When some transaction has committed and leaves the simulation system, a new transaction is generated and injected into the system to maintain the multiple programming level. The Pressing Mode is used in our parallel recovery simulation. It is easy to find out the maximum capacity of a system by increasing the MPL parameter, when the MPL parameter reaches a specific number, the system will encounter resource contention and/or data contention, the system throughput will not increase any more, and transaction response time gets worse, which indicates that the system is saturated.

Table 2. Parameters for Workload Modeling

Parameter	Meaning	Value
Update Transaction Percentage	Percentage of Update transactions	R20U80
Transaction Size	Number of tuples accessed	5-10
MPL	Multi Programming Level	128
Access Pattern	Access Pattern	80：20
Tuple Select Instructions	Cost for select a tuple	450
Tuple Update Instructions	Cost for update a tuple	450
Startup Instructions	Cost for start up an operation	1000
Terminate Instructions	Cost for terminate an operation	2000
Lock Instructions	Cost for Lock/Unlock	300
Latch Instructions	Cost for Latch/Unlatch	30
IO Instructions	Cost for an IO Operation	5000

The workloads in our simulation experiments are synthetic. However, we have captured fundamental characteristics of the above mentioned update intensive applications, and some parameters are used to model a range of contention, both data and resource. The performance of the proposed idea in a given real life environment could be determined and optimized using the results from the synthetic workloads.

3.5. Performance Metrics

We are concerned about two critical performance metrics.

(1) Transaction Response Time (millisecond)**:** Every transaction is assigned a startup time (absolute simulation time) when it is generated. After the transaction commits, we can compute its response time by subtracting the startup time from the commit time. All transactions' response times are registered so that a response time distribution diagram could be drawn. For update intensive main memory database applications, response time distributions carry more information than an average response time.

(2) System Throughput (transactions per second)**:** During simulation execution, the number of transactions executed is tracked. Using the number of transactions executed and total simulation time, average system throughput could be easily computed.

4. EXPERIMENTS

Three recovery schemes are simulated for comparison. Table 3 summarizes properties of the three schemes.

Table 3. Comparisons of the three Recovery Schemes

#	PLFC	DLTP	DLRP
	Physical Logging & Fuzzy Checkpoint	Differential Logging with Transaction based Log Partition	Differential Logging with Resource Based Log Partition
Logging	Physical	Differential	Differential
Checkpointing	Fuzzy	Fuzzy	Consistent
Log Partition	By Transaction ID	By Transaction ID	By Partition ID
Parallelism	Log ‖ Log	Log ‖ Log Data ‖ Log Data ‖ Data	Log ‖ Log Data ‖ Log Data ‖ Data
Log Scans	Two Passes (Undo / Redo)	One Pass	One Pass
Sys. Availability during Recovery	NO	NO	YES

The size of the database is 2GB, and log volumes of PLFC, DLTP, and DLRP are 721 MB, 377MB, and 373MB respectively (The number of transactions executed since last checkpoint is 600K), and the workload follows an 80:20 reference pattern.

4.1. Log Processing Time vs. Log Disk Number

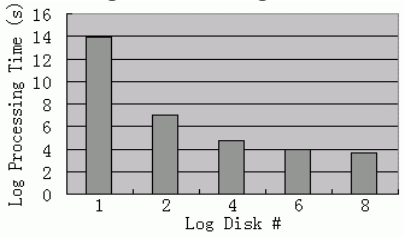


Figure 11. Log Processing Time vs. Log Disk Number

Since log records are read from log disks in parallel and applied to data partitions without the need of reordering, log-processing time is reduced. Figure 11 shows log processing times for various numbers of log disks (when the number of log disks is over 8, the IO channel is saturated).

Using multiple log disks could sharply reduce log-processing time.

4.2. Total Recovery Time

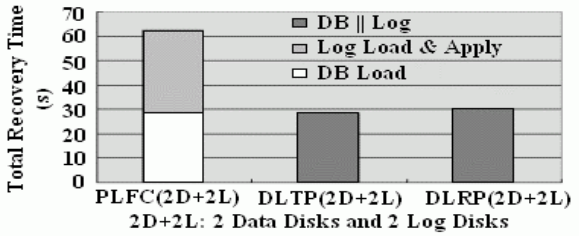


Figure 12. Total recovery time

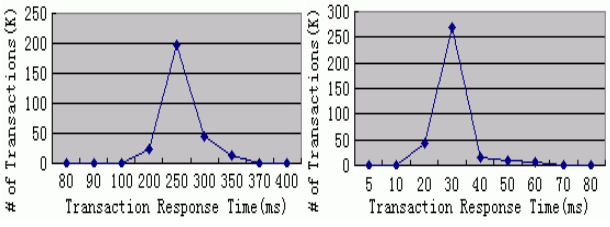
The total recovery time is broken down into checkpoint file loading time, and log processing time. In PLFC, checkpoint file loading and log processing are done in sequence. In DLTP or DLRP, the two tasks are done in parallel. The total recovery time is denoted with DB ||Log in figure 12. Two data disks and two log disks are used.

4.3. Some Discussion

The DLRP scheme is inferior to the DLTP scheme in terms of total recovery time. It is due to the reason that, DLRP recovers database systems based on data partition. It needs more disk seeks to find the respective data partition and related log records. Although DLRP takes more time to recover the whole database, it provides the critical feature of system availability during recovery. The situation will become better when high end solid-state disks, which have no mechanical components (no seek times), become widely used in high performance DBMS systems.

4.4. System Availability during Recovery

Experiments are conducted to demonstrate the nice feature of the parallel recovery scheme, system availability during recovery. Figure 13 shows transaction response time distributions (RTD) and the number of transactions executed during recovery. The recovery process is broken into four phases. At initial stages of recovery, the average response time of transactions is much longer than those of afterward stages due to the waiting for data partitions to be loaded and restored in main memory. The average transaction response time of the first quarter is 276 ms. When hot spot data is gradually rebuilt in main memory, transactions do not need to wait for IO operations to finish when accessing data partitions, the system executes transactions efficiently, and transaction response time sharply drops to 5.3 milliseconds.



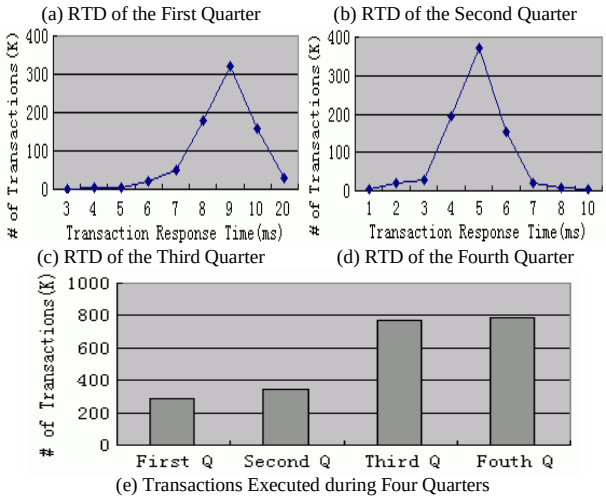


Figure 13. Transaction RTDs & Transactions Executed

5. RELATED WORKS

Juchang Lee, et al. have proposed a parallel recovery scheme for main memory DBMS, which is based on differential logging, in [10] [11] (DLTP Scheme). The recovery scheme takes minimal total recovery time, but it cannot provide system availability during recovery. New transactions have to wait until the entire database is recovered. The DLRP scheme presented in this paper employs a different log partition approach. It provides system availability during recovery.

Partition based checkpointing techniques have been studied [12], [13], [14] with the expectation that interference of checkpointing with normal transaction processing is reduced as well as checkpointing efficiency is improved. All of these checkpointing schemes fall into the fuzzy checkpointing category. To recover a database from a fuzzy checkpoint file, the log files have to be scanned for undoing of abort transactions and redoing of committed transactions. The DLRP scheme employs an efficient transaction consistent checkpointing scheme based on dynamic multi-versioning, which reduce interference of checkpointing with normal transaction processing, and simplifies recovery.

To reduce log-writing time and log processing time during recovery, redo only logging for main memory database systems has been studied [15]. Traditional redo only logging requires that the log records should be applied to data items in the serialization order so that the database is restored correctly after failure. The DLRP scheme is based on differential logging. Log records can be applied to a data partition in an arbitrary order without the need of reordering, which speedups recovery.

Levy and A. Silberschatz proposed log driven checkpointing as well as on demand recovery in [16], then Dongho Lee, and Haengrae Cho further improved it in [17]. On demand recovery provides almost immediate system availability during recovery. However, log-driven

checkpointing suffers from limited system throughput due to the intrinsic low efficiency of update propagation to checkpoint files [18]. The DLRP scheme not only provides system availability during recovery, but also cuts down the transaction's response time by parallel recovery.

6. CONCLUSION

A parallel recovery scheme for update intensive main memory database systems is presented. The scheme uses non-volatile memory to temporarily hold log records and decouple transaction committing from disk writes. Parallel properties of differential logging are exploited to improve logging efficiency by distributing log records to multi-log disks. The scheme employs a partition based consistent checkpoint technique, which simplifies recovery. During recovery, three types of parallelism are used to reduce the total recovery time. In the meantime, priority-based data partition recovery provides system availability during recovery. The database system can be devoted to new transaction handling soon after the meta-data is restored.

To evaluate the proposed idea, we built a simulation system using the JSIM simulation package and conducted a series of experiments. The experimental results show the effectiveness of the parallel recovery scheme. Simulation building and experimenting testifies to the power of J-SIM in complex software simulation.

References

[1] Stephen Blott, Henry F. Korth. An almost-serial protocol for transaction execution in main-memory database systems. In Proceedings of the 28th VLDB, 2002, pp.706-717.

[2] McObject LLC. In-Memory Database Systems Beyond the Terabyte Size Boundary. http://www.mcobject.com/130/EmbeddedDatabaseWhitePapers.htm, 2007.

[3] Intel Corporation. Intel Quad Core Xeon Processor Datasheet. http://download.intel.com/design/xeon/datashts/318589.pdf, 2008.

[4] Katsutaka Kimura, Takashi Kobayashi. Trends in High Density Flash Memory Technologies. IEEE Conference on Electron Devices and Solid State Circuits, 2003, pp. 45-50.

[5] MTRON Media Experts Group. MSD-P Series Production Specification. http://www.mtron.net/English/Customer/downloads.asp?sid=download&category=d02, 2007.

[6] Qin Xiongpai, Xiao Yanqin, Cao Wei, Wang Shan. A Parallel Recovery Scheme for Update Intensive Main Memory Database Systems. Proceedings of The 2008 International Conference on Parallel and Distributed Computing, Applications and Technologies, Dec., 2008, pp. 509-516.

[7] Paul M. Bober, Michael J. Carey. On Mixing Queries and Transactions via Multiversion Locking. In Proceedings of 8th ICDE, 1992, pp.535-545.

[8] James F. Power, John T. Waldron. Recent Advances in Java Technology: Theory, Application, Implementation, Chapter 3: Discrete-Time Process-Oriented Simulation with J-Sim (First Edition). ISBN 0-9544145-0-0, Computer Science Press, Trinity College Dublin, Dublin, Ireland, 2002, pp.21-30.

[9] Ole-Johan Dahl, BJorn Myhrhaug, Kristen Nygaard. Some features of the SIMULA 67 language. Proceedings of the second conference on Applications of simulations, New York, United States, 1968, pp.29-31.

[10] Juchang Lee, Kihong Kim, Sang K. Cha. Differential Logging: Acommutative and Associative Logging Scheme for High Parallel Main Memory Database. Proceedings of the 17th ICDE, 2001, pp.173-182.

[11] Sang K. Cha, Changbin Song. P*TIME: highly scalable OLTP DBMS for managing update-intensive stream workload. Proceedings of the Thirtieth VLDB, 2004, pp.1033-1040.

[12] J. L. Lin and M. H. Dunham. A performance study of dynamic segmented fuzzy checkpointing in memory resident databases. Technical Report, 96-CSE-14, Dept. of Computer Science and Engineering, Southern Methodist University, 1996.

[13] Jing Huang, Le Gruenwald. Crash recovery for real-time main memory database systems. In Proceedings of the 1996 ACM symposium on Applied Computing, 1996, pp.145-149.

[14] Liao Guoxiong, Liu Yunsheng, Xiao Yingyuan. A Partition Fuzzy Checkpointing Strategy for Real-Time Main Memory Databases. Journal of Computer Research and Development, Vol 43(7), 2006, pp.1291-1296.

[15] H.V.Jagadish, A.Silberschatz, and S.Sudarshan. Recovering from main memory lapses. In Proceedings of the 19th VLDB, 1993, pp.391-404.

[16] E.Levy and A.Silberschatz. Incremental recovery in main memory database systems. IEEE Transactions on Knowledge and Data Engineering, Vol 4(6), 1992, pp.529-540.

[17] Dongho Lee, Haengrae Cho. Checkpointing schemes for fast restart in main memory database systems. In IEEE Pacific Rim Conference Communications, Computers and Signal Processing, 1997, pp.663-668.

[18] Yutong Wang, Vijay Kumar. Performance comparison of main memory database recovery algorithms. In 8th International Conference on Management of Data (COMAD '97), 1997, pp.55-63.

Biography

QIN Xiongpai was born in 1971. He received the Bachelor degree in computer science from Renmin university of China in 1994, and received the Master degree in computer science from the same university in 2001. He is a Ph.D. candidate at Renmin University of China at present. He is also a lecturer at the university. His research interests include DBMS tuning, and high performance DBMS.

CAO Wei was born in 1975. She received the Bachelor degree in computer science from Renmin university of China in 1997, and received the Master degree in computer science from Information School, Renmin university of China in 2000. She is a Ph.D. candidate at Renmin University of China. She is also a teaching assistant at the university. Her research interests include DBMS tuning, and self-managing DBMS.

WANG Shan was born in 1944. She graduated from Peking University in 1968 and received a M.S. degree in Computer Science from Renmin University of China in 1981. She is currently a full professor and Ph.D. supervisor in School of Information, RUC. Her current research interests include high performance DBMS, data warehousing technology, and grid data management.