RECOVERY FOR MEMORY-RESIDENT DATABASE SYSTEMS

By

HWEI JIUN CHANG

Bachelor of Arts

Chinese Culture University

Taiwan

1981

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1988

RECOVERY FOR MEMORY-RESIDENT

DATABASE SYSTEMS

Thesis Approved:

D. E. Hedrick

Thesis Adviser

Huizhu Lu

Blayne E. Mayfield

Norman N. Durham

Dean of the Graduate College

PREFACE

This paper presents a recovery mechanism for memory-resident databases. It uses some stable memory and special hardware devices to eliminate expensive I/O operations handled by the main processor. And, through this achievement, the throughput rate is improved.

I wish to thank my major adviser, Dr. G. E. Hedrick, for his guidance and invaluable aid. Thanks as well to the other committee members, Dr. M. Folk and Dr. H. Lu, for their advisement of this paper.

Special thanks are due to Dr. D. D. Fisher, for his helpful suggestions at the beginning of this paper.

My deepest appreciation is extended to my parents and my hushand for their constant support, moral encouragement, and understanding during my coursework at Oklahoma State University.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

CHAPTER I

INTRODUCTION

When a database system does not perform according to its specifications, a failure occurs. A failure is an event which places the system into an error state. Some failures are caused by human errors; e.g., a user mounts a wrong disk, software faults; e.g., inappropriate data, or hardware faults; e.g., loss of power. When a system becomes inoperable, several problems must be addressed. First, normal functions must continue. Second, computer operational and maintenance personnel must work quickly to restore the system as closely as possible to the last non-failing state. Third, users must know what to do when the system becomes available again. Because some work may need to be re-entered, users must know how much work to repeat. In order to cope with failures, additional components and algorithms for abnormal situations are added to a database system. These components and algorithms both remove erroneous data and restore the database systems to correct states from which normal processing can continue. These additional components and recovery algorithms used to return to normal states from abnormal states in database systems are called recovery techniques.

Terminology

Database technology can seem complex and complicated. In part, this is because database terminology is inconsistent. Similar concepts have different names; for example, object and entity are synonyms in some contexts, and the same name often refers to different concepts; for example, the term object has different meanings depending its context. This situation exists because database technology does not originate from a single source [11]. Therefore, in this section, a general description of database terminology is given. The single terminology presented here is used throughout the paper.

A database consists of a collection of logical records. The record is the granule at which transaction interface operates. Records also are grouped into large units called pages and segments. If a page has a new update copy, then the old page is the shadow for the new one(the updated copy). The new page is called a dirty page. Pages are the granule of data transfer between the primary and secondary memory. A segment is a granule of storage organization in secondary storage.

A transaction is a linear sequence of actions with the following properties:

Atomicity: either all actions are done or nothing happens.

Consistency: the property of being able to change the overall logical and physical structure of the database when a transaction is completed. Thus, it preserves the consistency of the database.

Durability: a characteristic of a database in which once a

transaction is committed to the database, the results of the transaction survive any system failures.

Isolation: the condition of events within a transaction being hidden from other transactions which run concurrently.

A general example of a transaction which transfers money from one account to another is given [20] (Figure 1).

```
FUNDS_TRANSFER:PROCEDURE;
 $BEGIN_TRANSACTION;
 ON ERROR DO;                          /* in case of error */
  $RESTORE_TRANSACTION;                /* undo all work */
  GET INPUT MESSAGE;                   /* reacquire input */
  PUT MESSAGE('TRANSFER FAILED');      /* report failure */
  GO TO COMMIT;
 END;
 GET INPUT MESSAGE;                    /* get and parse input */
 EXTRACT ACCOUNT_DEBIT, ACCOUNT_CREDIT, AMOUNT
         FROM MESSAGE;
 $UPDATE ACCOUNTS                      /* do debit */
    SET BALANCE = BALANCE - AMOUNT
      WHERE ACCOUNTS.NUMBER = ACCOUNT_DEBIT;
 $UPDATE ACCOUNTS                      /* do credit */
    SET BALANCE = BALANCE + AMOUNT
      WHERE ACCOUNTS.NUMBER = ACCOUNT_CREDIT;
 $INSERT INTO HISTORY                  /* keep audit trail */
    (DATE,MESSAGE);
 PUT MESSAGE ('TRANSFER DONE');        /* report success */
 COMMIT:                               /* commit updates */
 $COMMIT_TRANSACTION;
END;                                   /* end of program */
```

Figure 1. Example of a Transaction

In the above example, a transaction is initiated explicitly when an existing process issues BEGIN_TRANSACTION. All changes made by the transaction are recorded in the transaction's logical file called the log. Two records usually are retained on the log. The first is a copy of every

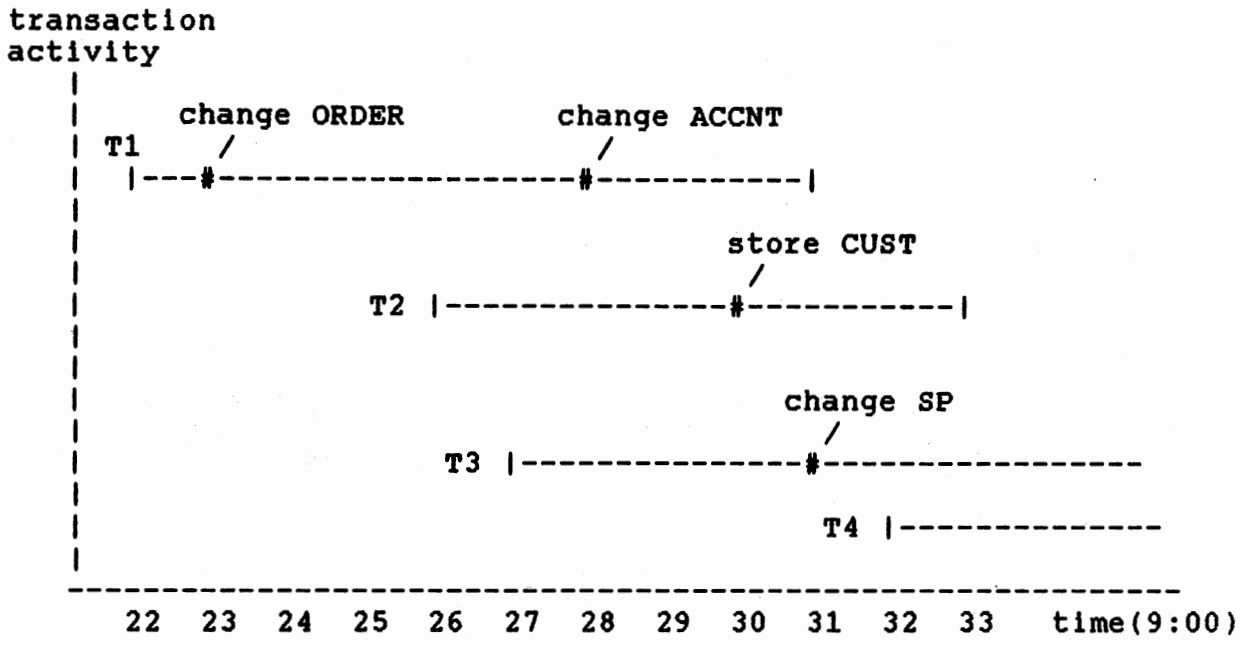
record before it was changed. Such records are called before images. The second is a copy of every record after it was changed. These records are called after images.

If at any point in time before reaching the COMMIT_ TRANSACTION something goes wrong, the user enters the ERROR clause and the update may be undone. If the transaction reaches the normal end but has not committed its results to the database, then it is always redone. This is the case when a system crash occurs.

There are two ways for a transaction to commit its results: first, the transaction flushes its own records to the log disk before completion; second, the transaction log records are not written to the log disk, instead, they are collected into a log page and a flush is delayed until a log page becomes full. Commits of the first sort are called immediate commits, those of the latter sort are called group commits.

The log itself is recorded on a dedicated medium. Once a log record is recorded, it cannot be updated. Figure 2-a shows an example of transaction activities happen at a time period. These activities are recorded on a transaction log (Figure 2-b).

For this sample log, each transaction has a unique name for identification purposes. Further, all images for a given transaction are linked together with two way pointers. One pointer points to the previous transaction-related record. The other pointer points to the next transaction-related record. A zero in the pointer field indicates the end of the list. The



a. Transaction Activity

relative record #

#								
1	T1	0	2	9:22	START			
2	T1	1	5	9:23	MODIFY	ORDER	old value	new value
3	T2	0	6	9:26	START			
4	T3	0	7	9:27	START			
5	T1	2	8	9:28	MODIFY	ACCNT	old value	new value
6	T2	3	10	9:30	INSERT	CUST	value	
7	T3	4	12	9:31	MODIFY	SP	old value	new value
8	T1	5	0	9:31	COMMIT			
9	T4	0	13	9:32	START			
10	T2	6	0	9:33	COMMIT			

b. Log Instance for Four Transactions

Figure 2. Transaction Log

recovery manager uses these pointers to locate all records for a particular transactions rapidly.

Other data items in the log are the time of the action, the type of operation (START, COMMIT, INSERT, MODIFY, etc.), the object being modified, and the before and after images. The before images are always written to the log before the change has been made to the database. This is known as the write ahead log protocol [16] so when a failure occurs after the log has been written, but before the database has been changed, all activities are known. In addition to these fields in the log, some other data items can be added, such as action identifier, length of log record, and record identifier.

Logs, sometimes called audit trails or journals, are used in the recovery process. Given a log with both before and after images, the undo and redo operations are straightforward. Undoing a transaction involves applying before images of all of its changes to the database. Redoing a transaction involves applying after images of all of its changes to the database. In this case, the before and after images are sometimes referred to undo-information and redo-information. This action assumes that an earlier version of the database is available. If it is necessary to restore a database to its most recent usable state and to reapply all transactions, then a great deal of processing time may be required. To minimize this problem, the database management system provides a facility called a checkpoint.

Checkpointing algorithms require the system periodically

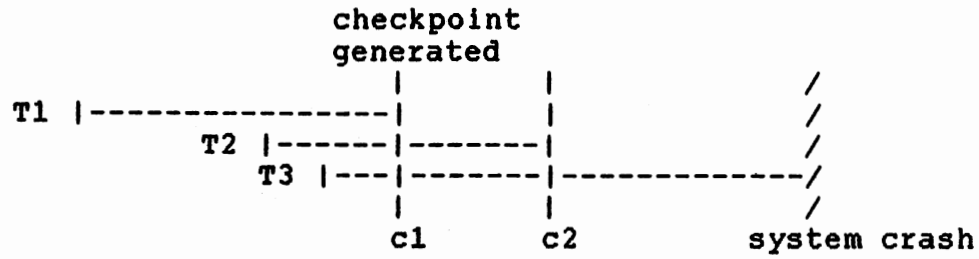
make a copy of the database. The checkpoint process consists of writing a BEGIN_CHECKPOINT record in the log, along with a list of currently active transactions, then flushing a backup copy of the database on secondary storage, and finally writing an END_CHECKPOINT record in the log.

Checkpointing is necessary for database recovery because it affects the amount of work that needs to be done at recovery time. Four distinct approaches are introduced [22] to show how checkpoint activities generated:

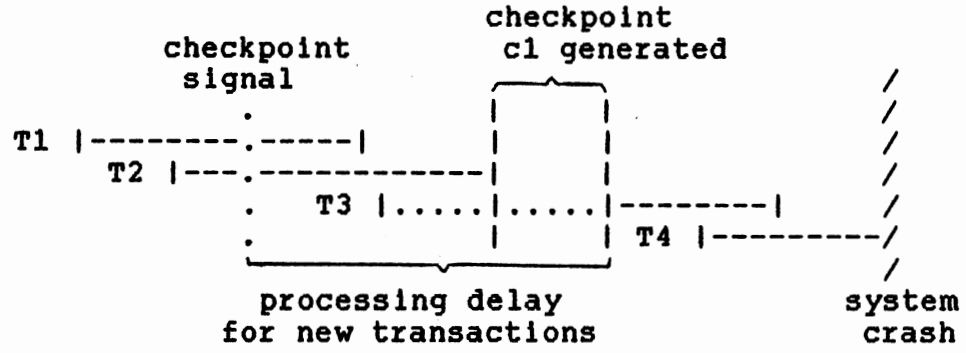
1. Fuzzy Checkpoints. The backup database is being produced while executing transactions are occurring. The backup database produced by such a checkpoint is called fuzzy because it may contain partial updates from transactions.

2. Transaction-Oriented Checkpoints. The checkpoint is initiated after a transaction is completed. Hence, the END_TRANSACTION record of each transaction can be interpreted as a BEGIN_CHECKPOINT and END_CHECKPOINT record. Transaction-oriented checkpoints are given in Figure 3-a. Checkpoints c1 and c2 are taken when transactions T1 and T2 reach normal termination.

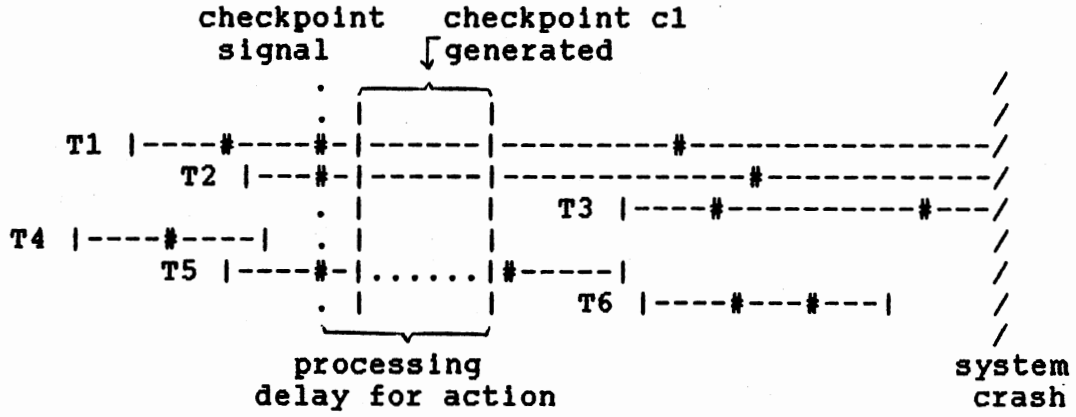
3. Transaction-Consistent Checkpoints. When a checkpoint generation is signaled by the recovery components, all incomplete transactions must be completed. Then the checkpoint is performed and all new transactions are delayed. After the END_CHECKPOINT record has been written to the log, normal processing is resumed. This is illustrated in Figure 3-b. Transactions T3 begins after checkpoint c1 is taken and



a. Transaction-Oriented Checkpoints



b. Transaction-Consistent Checkpoints



c. Action-Consistent Checkpoints

Figure 3. Three Different Criteria for Checkpoints

completes before the system crash, so T3 needs to be redone; whereas, transaction T4 is incomplete, so it must be undone. There is no effect on transactions T1 and T2, since their updates are saved on the checkpoint disk.

4. Action-Consistent Checkpoints. Action-consistent checkpoints are generated in a way similar to the transaction-consistent checkpoints. The checkpoint for action consistency is generated when no update action is being processed. Figure 3-c illustrates action-consistent checkpoints. The actions of transactions T1 and T2 since the preceding checkpoint, c1, must be undone. Transaction T3 must be rolled back. The recovery process must redo the last action of transaction T5 and all of transaction T6.

Checkpoints also are called dumps and/or saves. There are two different aspects of checkpoints. Either the entire database or only those portions of the database that have updated since the last checkpoint are recorded on each iteration. Checkpoints which belong to the former classes are called full checkpoints. The others are called partial checkpoints. Using the log together with a most recently checkpointed database, the recovery manager can restore the database to a usable state from which normal processing is allowed to proceed.

Types of Database Failures

A wide variety of failures can occur in processing a database, ranging from the input of incorrect data to complete loss or destruction of the database. Three of the most common

types of errors are aborted transactions, system failure, and database loss or destruction. Each of these types of errors is described below, and the most common recovery procedure is indicated.

Transaction Failure. For some reason, the transaction does not reach its normal termination. Example of such errors are deadlocks, timeout, incorrect input data, and protection violations.

When a transaction aborts or must be aborted by the system, any changes made by the transaction but not yet committed to the database must be undone in reverse order. The recovery action for this kind of failure is called transaction UNDO.

System Failure. The system is shut down in an uncontrolled manner. The contents of main storage are lost. Such a failure can be caused by an operating system fault, power loss, or operator error.

When the system crashes, the changes caused by all incomplete transactions must be removed, and the changes caused by all completed transactions must be redone. The recovery action of the first sort is called global UNDO; whereas, the latter is called partial REDO.

Media Failure. A media failure is a failure in which some portion of the database has been destroyed physically. A typical cause of media failure is a disk head crash.

A backup copy of the database is required for recovery in this situation. The first step is to restore the latest

consistent backup copy and then performs REDO operations for all transactions completed since the copy was created. This recovery action is called global REDO.

The above three types of failure are generally happen in classical database systems. In Haerder and Reuter [22], they give some interesting empirical figures regarding frequency of occurrence and typical recovery times for three kinds of failure in a typical large system (TABLE I).

TABLE I

FREQUENCY OF OCCURRENCE AND RECOVERY TIME FOR THREE TYPES OF FAILURE

Failure Type	Frequency of Occurrence	Recovery Time
Transaction	10 to 100 per minute	same as trans. execution time
System	Several per week	few minutes
Media	Once or twice per year	1 to 2 hours

Literature Review

Making computers easier to use is the goal of most software. Database management systems, in particular, provide a programming interface to ease the task of writing electronic bookkeeping programs. The recovery manager of such a system in turn eases the task of writing fault-tolerant application

programs [27] [30].

System R is a database system which provides a relational model of data. It uses write-ahead logging in combination with shadow pages [21] [31] to support COMMIT, ABORT, and UNDO actions. A major virtue of shadows is that they ensure that a system restart always begins with a RSS (Research Storage System. an internal system which supports data access method) action-consistent state. This is quite a simplication and probably contributes to the success of system restart. Shadow schemes, however, consume an inconsequential amount of disk space. On the other hand, in order to use the shadow mechanism, one must reserve a large amount of disk space to hold the shadow pages.

Database cache [14] is the other recovery mechanism for disk-based databases. It uses large amounts of main memory space to store all currently active pages plus some other pages which are needed for reading. The design of database cache is to achieve the goal of high throughput of short transactions. A long update transaction may cause the cache to overflow. A demand paging technique can be used to bring pages into main memory [31] to avoid the overhead of using the entire database. No log is used, rather a safe located in non-volatile memory containing data needed to reconstruct part of the cache after failure is maintained.

With the traditional databases, the current database state exists partially in main memory, and partially in secondary storage. Retrieval and update transactions suffer

the long delays caused by disk I/O when the desired record does not reside in primary memory. Due to the declining cost per bit of main memory and to rising chip densities, it is becoming feasible to store complete databases in primary memory. With the entire database in main storage, transactions suffer no disk delays. As a result, the memory-resident database system can improve performance through reduced CPU overhead as well as through the elimination of disk access time. Because of the volatility of main memory, main memory databases complicate database recovery issues, This makes the recovery operations for disk-based databases different from that for memory-resident databases.

When discussing the memory-resident database recovery, it is important to realize that any recovery schemes must deal with data in primary storage. Secondary storage is used only for backup purposes. In memory-resident databases, a system failure can be treated as a media failure with disk-based databases and a global REDO performed. Media failures with memory-resident databases can effect the archive database or log, restoring these files may mean a global REDO applied. However, when the specific location of media failure can be identified, a partial REDO is required to recover the affected area. The differences between disk-based database and memory-resident database recovery operations are listed in TABLE II.

The issues concerning memory-resident database recovery have been receiving increased exposure over the last few years. One of the first memory-resident databases is IMS/VS Fast Path

[24] [25]. IMS/VS Fast Path is the first commercial product that uses the idea of group commit to reduce traffic to the log disk by delaying flushes of several transactions' log records during the commit phase. Transactions must spend additional time waiting for their commit groups to assemble. This becomes a great influence on throughput.

TABLE II

DATABASE RECOVERY OPERATIONS

Failure Types	Recovery Operations: Traditional Databases	Recovery Operations: Memory-resident Databases
Transaction	Transaction UNDO	Transaction UNDO
System	Global UNDO Partial REDO	Global REDO
Media	Global REDO	Global REDO Partial REDO

DeWitt et al. [10] describe a recovery method with the possibility of stable memory. They use a small non-volatile random access memory as a log buffer to perform log compression through which some undo and redo items can be eliminated. They also proposed an overlapped checkpointing algorithm which requires a high degree of synchronization and data sharing.

Additional concerns center around the increasing in the number of main memory components. These concerns are under investigation at Princeton University. The Massive Memory Machine [15] project is designed to support massive amounts of primary storage to allow the serial execution of transactions. The improved performance can eliminate the need for concurrency control. Associated with the Princeton project is the design of a main memory database recovery scheme based on a hardware logging device, HALO [18]. HALO monitors the main CPU, intercepts word-level writes to the database, and logs them before passing them onto the database system.

In [23], Hagmann proposed using the existing recovery techniques of fuzzy dumps and log compression to provide a fast restart after a crash. His design concentrates on medium-size main memory databases (approximately 1 Gbyte) that have many small updates; e.g., debit/credit transactions.

Another recovery technique for main memory databases is presented by Eich in [12]. Eich in her paper describes an automatic checkpoint which runs on a separate recovery processor. In order to accomplish automatic checkpointing, the log manager monitors the log state and finds the most recent checkpoint record on the log, then the recovery processor waits for the database system to become quiescent and performs the checkpoint.

Putting the system into a quiescent state until no update transaction is active may cause an intolerable delay for incoming transactions. An algorithm for continuous consistent

checkpointing is presented by Pu in [32]. Pu states that the database system does not need to be quiesced to obtain a consistent checkpoint; instead, the checkpoint runs concurrently with the normal transaction processing, and locks the entities in the database one by one so that transactions which do not interfere with the checkpoint process are allowed to run.

The design for a memory-resident database system including data structures, query processing, and recovery technique has been proposed by Lehman [28] [29]. Recovery processing uses a stable log buffer as well as a special log processor to perform the checkpointing operation. The use of a log processor reduces the amount of logging work done by the main CPU. Thus, through decreasing CPU cost, a greater response time in logging is achieved. Finally, [16] [17] presents a a taxonomy of previous recovery policies on main memory databases based on the update, logging, checkpoint, and backup policies.

Overview of Thesis

This paper examines recovery techniques for both disk-based databases and memory-resident databases, identifies differences between the two, and proposes a memory-resident recovery technique. The paper outlines the recovery mechanisms used in the disk-based databases; sketches previous work on memory-resident database recovery; introduces a proposed new design; presents a comparison among recovery techniques for

main memory databases; and concludes by listing areas for future work.

CHAPTER II

RECOVERY FOR DISK-BASED DATABASES

The first recovery algorithm of interest is the one used in System R [21]. System R consists of an external layer called the Research Data System (RDS) and a completely internal layer called the Research Storage System (RSS). The external layer provides a relational data model and operations thereon. The RSS is a nonsymbolic record-at-a-time access method. The RSS provides actions on the object it implements. Each segment consists of a page table with pointers to the data pages. Associated with each pointer in the page table are three bits: a shadow bit, a cumulative shadow bit, and a long term shadow bit. When a segment is updated, its new value is put in a newly allocated page, and the current version of the page table is updated to point to the new page. The backup version remains unchanged. For each page that is updated, both the shadow bit and the cumulative shadow bit are set in the page table entry of the segment containing the page. When the current state of the segment is saved, the shadow bits are switched off, and the old pages of the backup version, having been replaced by the new versions from the current copy, are released. Checkpoints for all the segments are taken regularly in an RSS action-consistent state. This involves copying all

of the pages of all segments in the system for which the cumulative shadow bit is on. The long term checkpoint bits are used to make sure that subsequent saves do not release the page before the checkpointing algorithm has copied them. Implementors using this design suggest both that shadowing is a very expensive process, and that logging would probably sufficient in their system.

The TWIST algorithm devised by Reuter [34] is designed for fast UNDO recovery. It uses a shadow pages scheme, allocating two physical blocks for each database segment; that is, it contains the new state of a segment and its before image in secondary storage. In the TWIST algorithm, each segment is augumented with a bit indicating which of the two backup blocks of that segment is updated most recently. When a checkpoint begins, it is assigned a timestamp. During checkpointing, the segment is written to the least recently updated of its two backup blocks. The timestamp of the checkpoint also is stored with that flushed segment. When recovery proceeds, the two backups of each segment are read. The block with the larger timestamp is chosen and the segment in primary memory is restored from that block.

Next in the TWIST algorithm is database cache [14]. It is designed to replace the traditional buffer, and therefore, allows an efficient solution to low database traffic. The design consists of three components: the physical database, the cache, and the safe. The physical database contains exactly one version of each database page. The cache, a part

of main memory space, holds all the pages that are needed for reading or modifying of an active transaction. The safe which resides on disk is a backup memory used to protect the contents of the cache in case of a system failure. When a transaction wants to update a page, then the desired page is read from the database into the cache as the original if it is not in the cache; otherwise, it is modified and becomes a dirty version of that page. When a transaction reaches the commit phase, all corresponding dirty pages are written sequentially onto the safe; the changed pages are written back to database from the cache with update-in-place. This implies that a transaction-oriented checkpoint is taken after every transaction. If a transaction is aborted, or aborts itself, all pages belonging to that transaction simply are released in the cache; therefore, no I/O is required. Recovering the database after a crash is simple, only involving loading the safe back into the cache, then normal processing is allowed to resume. The database cache approach shows high throughput for short transactions. However, in the case of a long transaction the cache cannot hold all of its pages so some of them must be written to disk, thus requiring the use of UNDO log records and write-ahead logging protocols.

Finally, a survey of recovery techniques used in traditional database systems is given [35]. These recovery techniques, applied in different environments, provide different kinds of recovery for databases and restore them to a usable state. They are:

1. Recovery to a correct state (a database is in a correct state both if the information in it consists of the most recent copies of data put into the database by users and if it contains no data deleted by user).

2. Recovery to a correct state which existing at some moment in the past (i.e. a checkpoint).

3. Recovery to a possible previous state.

4. Recovery to a valid state (a database is in a valid state if its information is part of the information in a correct state).

5. Recovery to a consistent state (a database is in a consistent state if it is a valid state, and the information it holds satisfies the users' consistency constraints).

6. Providing crash resistence.

Techniques employed for different kinds of recovery are divided into seven categories:

1. Audit trail -- An audit trail records the sequence of actions performed on a file. It can be used for the purposes of crash recovery and backing out to restore the database to a correct state.

2. Backup/current version -- The files contain the previous/present values form a backup/current version of the database. Backup version can be used to restore files to a previous state. If it is together with current version, they are used to restore files to a checkpoint state.

3. Careful replacement -- When the update is performed, the copy of a component, which replaces the original, is kept

until after the replacement is made successfully. In other words, two copies exist only during update; otherwise, there is just one copy containing the current value. This makes the update or sequences of updates as safe as possible by reducing the chance of being left with an inconsistent copy or mutually inconsistent files. This technique is used to restore a state prior to update.

4. Differential files -- The main file remains unchanged. All changes that would be made to a main file are recorded in a differential file. The differential files regularly are merged with the main files. A differential file is a type of audit trail, yet the actual updates have not been made. The differential file can be used to restore the database to a valid state.

5. Incremental dumping -- Incremental dumping creates checkpoints for updated files. It copies updated files onto archive storage either after a job has finished or at regular intervals. Incremental dumping provides a facility of restoring all the files to their previous consistent state.

6. Multiple copies -- At least two copies of each file are kept. The different copies are identical except during update. If the number of copies is odd, then a majority having the same value is taken as the correct one. If there are two copies of a file, then a bit can be used to indicate "update-in-progress," while the state is inconsistent. This technique provides crash resistence.

7. Salvation program -- A salvation program is a last

resort, used if all other techniques fail. It cannot bring the database back to a previous state. It only rescues the information that is still recognizable.

Although the traditional recovery algorithms may perform correctly on a disk-oriented database, they might not perform satisfactorily on a memory-resident database. Therefore, several recovery algorithms for a memory-resident database have proposed to log and checkpoint the memory-resident database efficiently.

CHAPTER III

RECOVERY FOR MEMORY-RESIDENT DATABASES

One way to classify main memory database systems is according to the number of users they support. This classification can affect the recovery components of the system. In a single-user system with only one user at a time processing the database, data integrity is simpler to maintain since data recovery algorithms can be implemented more easily. In contrast, multi-user database systems are accessed concurrently by many users. The recovery in a multi-user system is much more complex than recovery in a single-user database system. Special precautions need to be taken to prevent data inconsistence. The following sections introduce existing recovery methods for single-user and multi-user database environments.

Single-User Database Systems

Single-user memory-resident databases most frequently are found on personal microcomputers. Reflex is a database manager product from Borland International Company [4]. The database in Reflex is an organized collection of records, in which information is entered. With the displaying and manipulation of the database, Reflex provides five different

views for users to show the same information. These five views are: Form, List, Graph, Crosstab, and Report. Since all five views arise from the same underlying database, changes made to one view instantly affect the others.

Additional features provided by Reflex include the Translate Program and the Export facility. The Reflex Translate Program converts files created with other programs to the Reflex format. Conversely, the Reflex Export converts Reflex files into a form readable by other programs.

The second generation of the Reflex database management system is Reflex Plus. It is designed to support larger record sizes (up to 4,080 characters per record for the APPLE Macintosh users).

Data-recoverable capabilities of both Reflex and Reflex Plus are at the record level. For internal recovery, undoing any deletion of records is accomplished by a second confirmation by the user so that the data is not erased when the user accidentally deletes a record. Restoring removed and replaced columns involves performing commands to restore their original states without affecting the database. For external recovery, a recovery program called Flexrec is used to recover corrupted data from a disk crash. If the damaged portion of the disk is recoverable, then the program outputs the diagnostics to a .doc file and outputs the data which still is recognizable to a .prn file. The .doc file contains information about master record address, the address and the length of the data records section, the names of the fields,

and errors. The .prn file along with the .doc file are used both to identify damaged data and to restore the database.

The IBM OS/2 Extended Edition (EE) [26] has a bulit-in database manager with support for Structured Query Language (SQL). The EE's database manager consists of an SQL-based relational database engine, called Data Services, and a front-end application for this engine called the Query Manager. The database engine (a collection of organized information including the database itself and catalogs and access plans for the database) also can be accessed by embedding SQL code into custom applications. Like other SQL database engines the EE database divides data into a series of relational tables, with rows (records) and columns (fields). The user can construct a VIEW of a database by SELECTing various columns and JOINing tables.

The EE database also includes a transaction management. The goal of transaction management is to ensure that a transaction is completed successfully even when a catastrophe occurs while the transaction is being processed. This problem is handled by a pair of functions called COMMIT and ROLLBACK. All transactions first are written to a buffer. After the transactions are completed, they are written (or committed) to the database files. If a problem occurs before they are completed, then the transactions are rolled back and executed over again. The COMMIT and ROLLBACK functions are performed automatically, but the user also can have explicit control over them.

Another feature of the database engine for transaction management is a recovery log. The log lists each new record and the record that was replaced. Also, the recovery log is written to disk before the database on the disk is updated. With the log, the user can reconstruct the database or complete the updating of the database.

In addition to its SQL capabilities, the database engine also includes a number of interesting utilities. The system's BACKUP and RESTORE utilities let the user do short, incremental backups, or restore the database to any prior condition.

Some other current single-user main memory databases and the backup and recovery facilities they provide are listed in Appendix A [5].

Multi-User Database Systems

The IBM IMS Fast Path [24] [25] is the first system that uses a memory database which is treated differently from the rest of the disk-oriented IMS database. Page updates are not performed until commit time. Log records are not flushed immediately upon commit, rather they are collected in a special database buffer with other committed records so that the cost of writing the log pages can be amortized over several transactions. IMS Fast Path performs transaction-consistent checkpoints which write entire database to the archive after system quiesced.

Researchers at the University of California at Berkeley

[10] were the first to use the notion of a pre-committed transaction. When a transaction commits, its commit record is placed in the log buffer to allow other conflicting transactions processing to begin. This accelerates the commit process by reducing the amount of time a transaction must wait until its log records are flushed. Commit processing actually completes when the log buffer has been flushed to disk. Frequent action-consistent checkpointing is made in parallel with transaction processing. The previous checkpointed pages are written to disk as a temporary log while the memory copies of the checkpointed pages are written in-place to the disk. Once all the checkpointed pages have been written to disk, the temporary and memory copies are released.

IBM's Office by Example (OBE) [2] uses a memory-resident database design including data structure representation and recovery techniques. It is assumed that every relation participating in a transaction is read once at the beginning of the transaction and, if modified, written to disk at the end. OBE uses shadow pages in stable storage. All new copies of modified relations are written to shadow areas on the archive database at commit time. Transaction-oriented checkpointing occurs continuously. The cost of flushing log data at the end of every transaction appears to be very high.

The hardware logging device, HALO [18], is designed to reduce some recovery duties handled by the main processor, such as initiating I/O to the log disks and copying to buffers. HALO has the internal registers and data and command paths both

to the main CPU and to primary memory. HALO intercepts communications between the processor and the memory to create a before-image and after-image log. HALO contains stable random access memory which implies that the log buffers need not be flushed before allowing transactions to commit. The before-image log data is needed to undo aborted transactions. Action-consistent checkpoints to the archive database occur continuously and in parallel with transaction processing by reading the entire main memory database and identifying changed pages.

Hagmann [23] outlines a method of doing recovery that uses fuzzy dumps and log compression to provide a quick transaction processing and rapid restart after a crash. A memory image is periodically written to disk while the normal database system is running and modifying the database. There is very little coordination between the dumper and the main database system. The dump is inconsistent since it may contain partial updates from transactions. This method provides an almost up-to-date and stable copy of the database with the oldest data only a few minutes old. In order to support this type of fuzzy dump, the UNDO and REDO log information must be in physical before-image or after-image form. Therefore, the log grows large quickly. Log compression then is needed to keep the log short so that crash recovery only processes a small amount of log; that is, the redo part of aborted transactions and the undo part of committed transactions can be eliminated. This is performed by a

software compressor. The logging, in this case, is done at the page level which requires much more log data to be manipulated than if logging were done at the record level.

Another technique that uses main memory shadow pages, pre-committed transactions, automatic checkpointing, and a recovery processor is given by Eich [12]. Main memory shadow pages are used to achieve the goal of no I/O for transaction UNDO. Update transactions create duplicate copies of these pages. At commit time, a commit record is written in the log buffer. As soon as this occurs, other conflicting transactions are allowed to progress. They use the data in the dirty pages or, if needed, create new copies of any modified pages. If a transaction commits, the previous clean pages are released and the dirty pages become the new clean ones. Undoing the effects of a transaction simply releases the dirty pages. The automatic checkpoint is obtained in a way that the log manager monitors the state of the log and keeps track of it. The log state has an initial value of 0. When the BEGIN_TRANSACTION record is written to the log, the state value is incremented by 1. Whereas the COMMIT_TRANSACTION and ABORT_TRANSACTION records decrement the state value by 1. When the log manager detects a state value of 0, a transaction-consistent checkpoint is then triggered. The checkpointing is accomplished by a recovery processor. It waits for the database system to become quiescent, then it blocks out other transactions while the entire database is written to disk.

Lehman [28] [29] sketches a method on efficient logging mechanism that uses stable random access memory and a recovery processor. Transaction update operations are performed in a volatile UNDO space at the record level. When a transaction terminates normally, records in the UNDO space are moved to the stable memory and become the REDO records. Transactions UNDO are done by discarding its UNDO records. The recovery manager, running on the recovery processor, organizes the REDO log records into partition bins. A partition bin is a unit of transfer that is larger than a typical disk page. As partition bins become full, they are written to the log disk. Each partition bin also has an update count. When a partition has accumulated a specified threshold count of log records, it is marked to be checkpointed. If a partition not having a sufficient number of updates but remaining in the stable memory longer enough, it is also marked to be checkpointed because of age. Actual checkpointing is performed by the transaction manager that runs on the main processor. For each partition checkpoint request, the transaction manager reads the specified partition from the database and writes it to the checkpoint disk on a transaction-consistent state basis. Since each partition is checkpointed at a time, the cost of a checkpointing is amortized over several update transactions.

CHAPTER IV

A PROPOSED RECOVERY DESIGN

Objectives

In most modern computers, the main processor is considered such a valuable resource that it should spend as little time as possible perform activities other than normal processing. In a high-performance database system, transaction throughput is important, so the time required for the commit phase should be small. This can be accomplished by using stable random access memory. In addition, any I/O needed should be performed asynchronously with normal processing. This implies that log I/O occur not only at commit time, but also throughout transaction processing. Moreover, frequent checkpoints are necessary to speed recovery in order to reduce the amount of log data that must be scanned. Checkpoint policies can be divided into two aspects: full and partial. For better efficient checkpointing algorithms, partial checkpointing is performed with little interference on transaction processing. Therefore, the use of three different processors to perform three tasks for normal processing, logging, and checkpointing is proposed to achieve the following requirements:

1. Reduce main CPU overhead.
2. Accomplish frequent checkpoint with little

interference on normal processing.

3. Acquire greater throughput.

Description of Hardware Components

The proposed model (Figure 4) is composed of a main processor, log processor, a recovery processor, stable memory, and a set of disks. The main processor -- an IBM 3090 model type of CPU -- is designed for heavy transaction processing loads which can process up to 79 millions of instructions per second (MIPS) [7] [8]. The VAX 8820, containing two VAX 8700 CPUs which perform as the log and recovery processing, offers performance of 11.4 MIPS [6]. Each processor in the VAX 8820 also can initiate its own I/O.

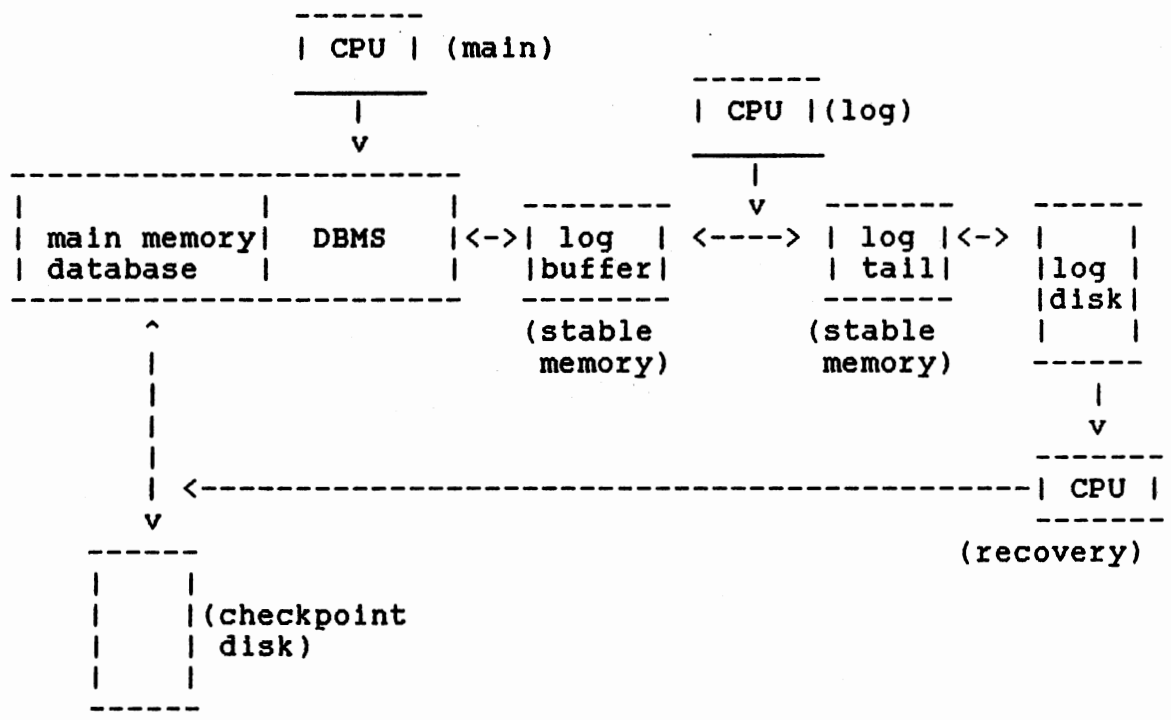


Figure 4. The Proposed Recovery Model

Furthermore, two to six VAXBI buses (an electronic link for input and output in VAX machines) are available on the VAX 8820 to speed the transfer between the main memory and secondary memory.

The recovery processor has access to the stable memory. The stable memory, made of non-volatile random access memory, is divided into two parts: a log buffer, and a log tail. The stable log buffer is used to hold transaction log records; whereas, the stable log tail contains units of log records. The disks used for maintaining recovery information are separated into two groups. One set of disks holds log information while another set of disks holds checkpoint information. Redundant copies of the recovery information can be provided to further protect data on secondary storage from a media failure.

The three processors have logically different functions. The main processor is in charge of regular transaction processing. The log processor manages the log information. It collects log records and groups them into units for transferring to the log disk. The two CPUs are required to shared only the stable log buffer, using it as a communication buffer along with its other uses. The recovery processor manages checkpointing operations, archive storage, and if necessary, restore the database in case of a system crash.

Logging

The logging procedure consists of three steps. First,

transactions create REDO log records. The REDO log records are placed in the stable log buffer. Second, the log manager, which runs on the log processor, reads the log records of committed transactions from the stable log buffer and places them into the stable log tail. In the stable log tail, log records are grouped into pages units. Third, these page units are written to disk when they become full.

When a transaction reaches its commit processing phase, the main CPU places its REDO log records in the stable memory so that the transaction can commit immediately. Once this is done, other conflicting transactions can begin to proceed. This is the only logging operation which involves the main CPU.

Since the REDO log records are kept in the stable memory, the log only maintains the after images of modified data. If a system failure occurs, then committed transactions are redone. If a transaction abnormally terminates, then undoing the effects of the transaction simply releases the UNDO records in main memory.

The log processor collects transaction log records in the stable log tail and organizes them into page units according to their corresponding memory allocations. When the log records fill up a log page, the records are ready to be written out to the log disk. The log processor initiates a disk write request for that page. Log records in that page unit are maintained in commit order so that they can sent to disk in commit order.

Checkpointing

System checkpoints are triggered at regular time intervals by model parameters. The recovery processor performs checkpoints at each interval to obtain an up-to-date backup copy of the database. Each database page is augumented with a dirty bit which is set by transaction updates and cleared when the page has been checkpointed. When a checkpoint begins, the checkpointing algorithm writes a BEGIN_CHECKPOINT record in the log, and scans through the database from the most recently checkpointed page until a dirty page is found. After a page has been identified as being dirty, the checkpointing algorithm sets a read lock on the page and waits in a high priority basis until it is granted (if there are several read locks in a waiting list, then the read lock issued by the checkpointing algorithm has the highest priority among others). When the read lock on that page is granted, the checkpointing algorithm allocates a block of memory large enough to hold the page, copies it into that memory, and released the read lock. Pages locks are held just long enough to copy at memory speeds, so there is little synchronization between the checkpointing and normal transaction processing. A checkpoint ends after writing an END_CHECKPOINT record in the log. Finally, the recovery processor records the address of the most recent checkpoint in the archive database.

The checkpoint disk space must be large enough to hold two

complete copies of the database: a previous copy and a current copy. The two backup copies are written alternately. This backup policy is a way to protect the archive database from a media failure.

Crash Recovery

Since the primary copy of the database is memory-resident, a transaction can begin to run if the information it needs is in main memory. Restoring the memory copy of the database involves reloading the most recent copy of the database, then using the log -- both active and archive portions -- to redo all transactions that completed since that copy was taken. There is no need to undo transactions that were still in progress at the time of the crash, since all updates of such transactions have been lost.

System restart proceed as follows: The recovery manager, running on the recovery processor, loads an earlier copy of the database back into the main memory. Next, it reads the log backwards to the point where the last checkpoint was taken. Then the database is rolled forward reapplying after images for all transactions that were proceeded after that checkpoint. Once the information has been restored, regular transaction processing begins.

Discussion of the Proposed Model

This section uses a performance model based on the model introduced in [1] [13] [33] to compare recovery methods in a

normal database system. Performance measures of transaction cost and throughput are derived based on estimate of CPU and I/O costs involved in database processing. The transaction cost includes costs for the main CPU and any I/O needed prior to commit processing. The transaction cost also includes costs for transaction undo and logging. The throughput rate is calculated by using the transaction cost. The checkpoint cost also is considered when calculating the throughput rate by reducing the main CPU processing power. The parameters and measures used by the performance model are shown in TABLE III.

Based on the Reuter's information [33], CPU time for accessing a page, Cp, and for copying a page, Cco, are taken to be 0.8 ms. Based on Reuter's statistics [33], the I/O time for writing a log record is assumed to be 10 ms. The Pd parameter is estimated half duplication of the modified pages which is required for checkpointing. Based on the information given by Eich [13], the probability of update transactions, Fu, is normally 0.25 and the percent of transaction undo is 0.03. Based on the statistics given by Agrawal and DeWitt [1], the percent of referenced pages of update transaction is 0.5. Based on the Reuter's information [33], the S parameter is the average number of pages referenced per transaction. It is taken to be 500. The main CPU involves the amount of logging activity is based on the size of a log record, Sr, since the size of a log record may influence on I/O. The Sr has a default value of 0.25. When a group commit is used, the number of transactions committed in a group is assumed to be

5. Based on [19], the checkpoint interval, I, is taken to be 300 seconds. Based on [22], the average time between system failures T, is assumed to be 3 days.

TABLE III

PARAMETERS AND MEASURES

Parameter	Description	Default
Cp	CPU time to access page	0.8 ms
Cco	CPU time to copy page	0.8 ms
Cio	I/O time to write page	10 ms
Pd	Percent of duplicate updates	0.5
Fu	Percent of update transactions	0.25
Pb	Percent of transaction undo	0.03
Pt	Percent of page updated	0.5
S	Number of page referenced	500
Sr	Size of log record	0.25
n	Number committed in group	5
I	Checkpoint interval	300 sec
T	Failures interval	3 days

Measure	Description
Cb	Cost for transaction undo
Cc	Checkpoint Cost
Cl	Logging Cost
Cr	Cost of retrieval transaction
Ct	Cost of average transaction
Cu	Cost of update transaction
Rt	Throughput rate

General types of transactions are transaction reads and writes. Most of the time the transactions are reads. The transaction cost is calculated based on the frequency of retrievals and updates:

$$Ct = Cr * (1 - Fu) + Cu * Fu.$$

In turn, the cost of retrieval transactions is based on the number of pages referenced:

$$Cr = S * Cp.$$

The cost updating a transaction includes the number of pages to be read, the cost for logging the updates of the transaction and the cost for transaction undo:

$$Cu = S * Cp + Cl + Pb * Cb;$$

where, Cb is the cost for rolling back the transaction in case of an isolated transaction failure, and Pb is the probability of such an event. Backout cost, Cb, is based on that no I/O is required and only a memory copy is needed:

$$Cb = Sr * (S * Pt) * Cco.$$

Computing the cost to perform logging has several different aspects. If special logging hardware is used, there is no impact on transaction processing so that Cl is taken to be 0. Without logging hardware, the calculations must consider writing one BEGIN_TRANSACTION and one END_TRANSACTION record per update transaction and, for each page being modified, a before_image and an after_image are needed. If a group commit is used, then log records are grouped together until a log page becomes full thus amortizing the I/O time over all update transactions in the group:

$$Cl = Sr * (2 * S * Pt + 2) * (Cco + Cio).$$

If an immediate commit is used, then each transaction flushes its log records before completing; thus, partial pages may be written:

$$Cl = (Sr * (2*S*Pt + 2) * Cco) + (\lceil Sr * (2*S*Pt + 2) \rceil * Cio).$$

Another calculation for Cl is based on that some stable memory is used. No I/O is needed before transaction commit:

$$Cl = Sr * (2*S*Pt + 2) * Cco.$$

The transaction cost is modified when a group commit is used. Since there are n transactions committed in a group, the value of Ct becomes:

$$Ct = (\sum_{i=0}^{n} i * Ct) / n.$$

The transaction throughput is obtained by reducing any checkpointing overhead provided by the main CPU. The Cc value is computed by using the Ct value obtained, determining the number of transctions processed between checkpoints, and then the number of updates pages:

$$Cc = (I * Fu * S * Pt * Pd * Cco) / Ct.$$

If checkpointing is performed by a separate processor, then there is no influence on the normal transaction processing so that Cc is assumed to be 0. The checkpoint cost is then used to determine the throughput rate:

$$Rt = (T * (1 - Cc/I) + Cc/2) / Ct / I.$$

This throughput rate is based on the assumption that the final crash occurs in the middle of a checkpoint interval.

Using this performance model, the throughput rate of various recovery techniques for memory-resident databases is shown in TABLE IV.

TABLE IV

THROUGHPUT RATE

Recovery Techniques	Throughput(transactions/sec)
IMS Fast Path	1.3
OBE	1.67
DeWitt	1.3
HALO	2.34
Hagmann	2.35
Eich	1.75
Lehman	2.27
Chang	2.42

The results based on throughput rate show several aspects which can affect the performance of transaction processing. First, the nonstable classes of recovery techniques give lower degree of throughput rate than the stable classes of recovery techniques. This is because, without stable memory, transaction commit processing cannot begin until all log I/O has been performed successfully. This I/O overhead directly affects response time. With enough stable memory to contain the log buffer, once log records have been written to the buffer commit processing can occur. Therefore, the use of stable memory eliminates the impact that logging I/O has on transaction performance. Second, transaction throughput is sensitive to the impact

of main CPU overhead for logging. If a logging device is used to perform the main CPU logging function, then the use of this hardware eliminates the main CPU overhead for logging.

Third, another impact on the throughput rate is checkpoint overhead. The use of a separate checkpoint processor can eliminate the main CPU for checkpointing which implies that the main CPU processing speeds increase. These factors show the availability of stable memory and special logging and checkpointing hardware are the crucial recovery factors impacting transaction throughput. Judging from these results, the proposed recovery algorithm gives a greater performance on transaction processing than prior recovery algorithms.

CHAPTER V

SUMMARY, CONCLUSIONS AND FUTURE WORK

Summary

In a high-performance memory-resident database system, transaction throughput rate is important. Such a system needs an efficient logging mechanism that can assimilate log records as fast as possible. It needs efficient checkpoint operations that can produce a reliable backup database and at the same time with little impact on normal processing. Finally, the recovery algorithm should not burden the main processor and affect transaction performance.

The new design for a recovery algorithm meets these three criteria. With the use of stable memory and a log processor, the logging mechanism cannot inhibit the performance of the system. Checkpointing operations are performed by a recovery processor, so very little synchronization is needed between the checkpointing processor and the main processor. With the ease of the tasks for logging and checkpointing, the main processor can work solely on transaction processing. After a crash, information requested by transactions are recovered first so that transactions processing can begin.

Partial memory recovery also in beneficial in the

event of a failure. When part of the main memory fails, the information in that part of memory is lost. Some existing recovery methods would have to recover the entire database. In the case of a partial memory loss with this proposal, only the lost portion of the database can be restored once the required information has been filtered out from the checkpoint copy and the log.

Conclusions and Future Work

Recovery techniques are used to ensure that any erroneous database state due to transaction, system, or media failure can be repaired to restore the database into a usable state from which normal processing can resume. Such techniques are used widely in disk-based database systems. However, some problems which exist in traditional database systems may not appear in a memory-resident database environment. The major problem of memory-resident databases deals with the volatility of main memory. This problem has been recognized and several new techniques for memory-resident database recovery have been proposed. Along with the useful ideas that have been generated so far, there are still several aspects of memory-resident database recovery that need better, more efficient algorithms.

This paper proposes a recovery technique for a memory-resident database system. A performance model for comparing different recovery techniques with respect to their impact on overall system performance is given. It shows that

some key parameters can influence database performance, and that the proposed recovery design meets these requirements better than previous methods. The performance model, however, is not intended to yield exact performance predictions in terms of throughput, rather it is intended to show three keys parameters -- stable memory, separate logging and checkpointing hardware -- influencing database performance most significantly. In order to approach an exact model, one must include many details about the implementation of the recovery algorithm and other components of the database management system it has to cooperate with. Simulation or improving the precision of the model is needed to determine how the various logging, checkpointing hardware and recovery operations interact when all three operations are running in separate processors. This is an area for future work.

SELECTED BIBLIOGRAPHY

[1] Agrawal, R. and DeWitt, D. J. "Integrated Concurrency Control and Recovery Mechanisms: Design and Performance Evaluation," ACM Trans. on Database Sys., 10, 4, (Dec. 1985), 529-564.

[2] Ammann, A., Hanrahan, M. and Krishnamurthy, R. "Design of a Memory Resident DBMS," Proc. IEEE COMPCON, San Francisco, (Feb. 1985).

[3] Bernstein, P. A. and Moodman, N. "Timestamp-based Algorithms for Concurrency Control in Distributed Database Systems," Proc. 6th International Conference on Very Large Database, (Oct. 1980).

[4] Borland International Inc. 4585 Scotts Valley Dr., Scotts Valley, CA 95066.

[5] ComputerWorld, "Data Dispersal Starts as Trickle," (March 14, 1988), S1-S12.

[6] ComputerWorld, "VAXs Tuned for Mainframe Challenge," (March 14, 1988), 1.

[7] ComputerWorld, "IBM Propels DB2 into Database Top Spot," (April 25, 1988).

[8] ComputerWorld, "Amdahl Tops IBM MIPS," (May 9, 1988), 1.

[9] Date, C. J. An Introduction to Database Systems 4th ed. Addison-Wesley Publishing Company, 1986.

[10] DeWitt D. J., Kate, R. H., Olken, F., Shapiro, L., Stonebraker, M. and Wood, D. "Implementation Techniques for Main Memory Database Systems," ACM, (1984).

[11] Dolan, K. A. and Kroenke, D. M. Database Processing: fundamentals, design, implementations , 3rd ed. SRA,1988.

[12] Eich, M. H. "Main Memory Database Recovery," Proc. ACM-IEEE Fall Joint Computer Conference, (1986).

[13] Eich, M. H. "A Classification and Comparison of Main Memory Database Recovery Techniques," Proc. 3rd International Conference on Data Engineering, Los Angeles, CA, (Feb. 1987), 332-339.

[14] Elhardt, K. and Bayer, R. "A Database Cache for High Performance and Fast Restart in Database Systems," ACM Trans. on Database Sys., 9, 4, (Dec. 1984), 503-525.

[15] Garcia-Molina, H., Lipton, R. J. amd Valdes, J. "A Massive Memory Machine," IEEE Trans. on Computers, C-33, 5, (May 1984), 391-399.

[16] Garcia-Molina, H. and Salem, K. "Checkpointing Memory-Resident Databases," Princeton University Computer Sciences Department Technical Report, December, 1987.

[17] Garcia-Molina, H. and Salem, K. "Crash Recovery for Memory-Resident Databases," Princeton University Computer Sciences Department Technical Report, November, 1987.

[18] Garcia-Molina, H. and Salem, K. "Crash Recovery Mechanisms for Main Storage Database Systems," Princeton University Computer Sciences Department Tech. Rep. April, 1986.

[19] Gray, J. "Notes on Data Base Operating Systems," in Operating Systems: an Advanced Course, G. Seegmuller, Springer-Verlag, (1978), 393-481.

[20] Gray, J. "The Transaction Concept:virtues and limitations," Proc. 7th International Conference on Very Large Databases, (Sep. 1981).

[21] Gray, J., McJones, P., Blasgen, M. Lindsay, B. Lorie, R., Price, T., Putzolu, F. and Traiger, I. "The Recovery Manager of the System R Database Manager," Computing Surveys, 13, 2, (June 1981), 223-242.

[22] Haerder, T. and Reuter, A. "Principles of Transaction-oriented Database Recovery," Computing Surveys, 15, 4, (Dec. 1983), 287-317.

[23] Hagmann, R. "A Crash Recovery Scheme for a Memory-Resident Database System," IEEE Trans. on Computers, C-35, 9, (Sep. 1986), 839-843.

[24] IBM, IMS/VS Version 1 FastPath Feature General Information Manual, GH20-9069-2, April 1978.

[25] IBM World Trade Systems Centers, IMS Version 1 Release 1.5 FastPath Feature Description and Design Guide, G320-5775, 1979.

[26] IBM Corp. Old Orchard Rd., Armonk, NY 10504, IBM's OS/2 Extended Edition.

[27] Lampson, B. and Sturgis, H. "Crash Recovery in a Distributed Data Storage System," XEROX Research Report, Palo Alto, CA, 1979.

[28] Lehman, T. J. "Design and Performance Evaluation of a Main Memory Relational Database System," CS Tech. Rep. #656, Computer Sciences Department, University of Wisconsin, Madison, WI, Aug. 1986.

[29] Lehman, T. J. and Carey, M. J. "Query Processing in Main Memory Database management Systems," Proc. of the ACM-SIGMOD International Conference on Management of Data, (May 1986).

[30] Lindsay, B., Selinger, P., Galtieri, C., Gray, J., Lorie, R., Price, T., Putzolu, F., Traiger, I. and Wade, B. "Notes on Distributed Databases," IBM Research Rep. RJ 2571, San Jose, CA, 1979.

[31] Lorie, R. A. "Physical Integrity in a Large Segmented Database," ACM Trans. on Database Sys., 2, 1, (March 1977), 91-104.

[32] Pu, C. "On-the-fly, incremental, Consistent Reading of Entire Databases," Proc. International Conference on Very Large Databases, Stockholm, (1985), 369-375.

[33] Reuter, A. "Performance Analysis of Recovery Techniques," ACM Trans. on Database Sys., 9, 4, (Dec. 1984), 526-559.

[34] Reuter, A. "A Fast Transaction-oriented logging Scheme for UNDO Recovery," IEEE Trans. on Software Engineering, SE-6, 4, (July 1980), 348-356.

[35] Verhofstad, J. S. M. "Recovery Techniques for Database Systems," Computing Surveys, 10, 2, (June 1978), 168-195.

APPENDICES

APPENDIX A

SINGLE-USER MAIN MEMORY DATABASES AND THEIR RECOVERY POLICIES

COMPANY	PRODUCT	BACK AND RECOVERY
Acius Inc. (408)252-4444	4th Dimension/ 4-D runtime	User implemented
AD & P Analysis, Design & Programming (703)790-9433	Ultra-base	Not provided
Advanced Business Microsystems Inc. (415)689-4515	Data ace	Backup, restore partial, entire database
Advanced Data Institute Inc. (916)381-8334	Aladin	Import,export,restart
Ashton-Tate Corp. (213)329-8000	Dbase III Plus	None
	Dbase IV	Full transaction processes, rollback,rollforward
Blyth Software Inc (415)571-0222	Omnis III Plus	None
	Omnis Quartz	None
Borland Internatinal Inc. (800)543-7543	Reflex	Media recovery
	Paradox 2.0	Table recovery
	Paradox 386	Table recovery
Brock Software Products Inc. (815)459-4210	Brock Keystroke relational DB	Rebuild function, floppy backup
Campus America Inc.(615)523-9506	Poise DMS-Plus	None
Century Analysis Inc.(415)680-7800	CFMS	Event rollback
Chang Laboratories (800)972-8800	C.A.T.	None

COMPANY	PRODUCT	BACKUP AND RECOVERY
Conceptual Software.(713)667-4222	Prodas	Backup copies of files
Condor Computer Corp.(313)971-8880	Condor 3 release 2.20	Automatic audit trails
Dataease International Inc. (203)374-8000	Dataease	Import,export;backup, restore of database
Empress Software Inc.(416)922-1743	Empress with m-builder	Transaction logging,warm restart, backup recovery
1st Desk Systems (800)522-2286	1stFile	None
	1stFile 4.0	None
	1stTeam	File copy
Fox Software Inc. (419)874-0162	Foxbase+	None
General Data Sys. (215)985-1780	GDX	Rollforward,rollback, warm start
IBM contact local sales office	OS/2EE	Full,selected backup; restore to state of last backup;load,unload table output;warm restart; commit,rollback functions
Infocom Inc. (617)576-1851	Cornerstone	Backup,restore
Informix Software Inc.(415)322-4100	Informix SQL	Transaction logs
	Informix 4GL	Transaction logs
Macon Systems Inc. (719)520-1555	ADBM	Internal backup, damaged file recovery

COMPANY	PRODUCT	BACKUP AND RECOVERY
MDBS Inc. (800)344-5832	Knowledgeman/2	Not provided
Microrim Inc. (206)885-2000	R:base	Load,unload,reload,verify integrity of database
Nantucket Corp. (213)390-7923	Clipper	File copy
	McMax	Backup, restore programs via programming language
Novell Inc. (512)346-8380	XQL	Proprietary
Odesta Corp. (800)323-5423	Helix VMX	Logging,autosave, save as revert to previous save
	Double HelixII	Same as Helix VMX
Oracle Corp. (800)345-DBMS	Oracle	Rollforward,rollback recovery
Prime Computer Inc.(617)655-8000	Prime Information	On-line transaction logging,rollforward, tape or disk backup.
	Prime Oracle	Rollback after image journaling;dynamic or static creation at AI files;import,export
Progress Software Corp.(714)969-2431	Progress	Crash-proof database engine,before image filing,rollforward,backup
Provue Development(714)969-2431	Overvue	Hard-disk backup
Relational Technology(800)4-INGRES	Ingres	Checkpoint,journaling, rollback,rollforward

COMPANY	PRODUCT	BACKUP AND RECOVERY
Rim Technology (206)451-8144	RTI Rim	None
Smith, Abbott& Co. (301)561-8411	Autopro	Audit trails, change logging
Software AG of North America (703)860-5050	Adabas	Save, restore; walk forward,backup functions
The Software Group (518)877-8600	Enable	Automatic database backup,restore
Sybase Inc. (415)548-4500	Sybase System	Physical logging;multiple database support;controllable guaranteed recovery time;log recovery;bulk copy program;journaling; table generator;monitoring tools;resource control;maintenance tool; consistency checker
	Dataserver	Same as Sybase System
	Datatoolset	Same as Sybase System
Unify Corp. (916)920-5553	Accell IDS	Transaction logging, database backup, rollforward recovery
	Unify relational DBMS	Same as Accell IDS
Wordperfect Corp. (801)227-500	Dataperfect	Regenerates indexes, file copy

APPENDIX B

GLOSSARY

Abort. To terminate a transaction abnormally.

AFIM. After image.

After image(AFIM). The new value of the updated item.

Atomic. An adjective describing the actions of a transaction that either are reflected in the database or nothing are happened.

Audit trail. See log.

BFIM. Before image.

Before image(BFIM). The previous value of the updated item.

Checkpoint(n.). A backup copy of the database.

Checkpoint(v.). Write the database to the backup disk.

Commit. A transaction which reaches normal termination never to be undone.

Dirty page. A page which is modified by a transaction that has not been committed yet.

Dump(n. or v.). See checkpoint.

Durable. An adjective describing the results of a committed transaction which must survive any malfunctions.

Full-checkpoint. The entire database which is written to disk.

Fuzzy dump(n.). A backup copy of the database which contains partial updates from transactions.

Fuzzy dump(v.). Copy the database in parallel with normal processing.

Global REDO. An operation for restoring the state of the database after it is physically destroyed.

Global UNDO. A procedure for removing the effects of any interrupted transactions from a system failure.

Group commit. Transactions whose records are contained on a log page which is not flushed to disk until it is full.

HALO. HArdware LOgging. A device used to perform logging functions.

Immediate commit. A transaction whose log records are flushed to the log before completing.

IMS. Information Management System. A transaction processing-oriented communications processor and DBMS developed by IBM.

Journal. See log.

Log. A logical file which contains information about active transactions.

Log compression. A process for a log which filters out any committed or aborted items since the latest checkpoint.

Partial-checkpoint. The portion of the database that have been updated recorded on a secondary device since the last checkpoint.

Partial REDO. A procedure for restoring the results of any completed transaction which may not yet reflected in the database after a system failure.

RDS. Research Data System. An external system of System R which supports the relational data model and the relational language SQL.

REDO. An operation for repeating the actions of a completed transaction from a system crash.

REDO-information. See after image.

RSS. Research Storage System. An internal system of System R which provides data access method.

Safe. A non-volatile memory used to protect the contents of the cache against loss.

Shadow page. An old page which is the shadow for the new one.

Stability. Non-volatility.

Stable log buffer. A stabe memory which keeps REDO log records.

Stable log tail. A stable storage where log records are grouped according to their correspoinding partition.

Stable memory. Non-volatile RAM.

Transaction. A sequence of actions.

Transaction UNDO. A procedure for recovery after a transaction failure.

UNDO. An operation for removing all effects of an incomplete transaction from a system failure.

UNDO-information. See before image.

Update-in-place. A performace which writes pages to the same block.

WAL. Write ahead log.

Write-ahead log(WAL). A log protocol which requires UNDO-information be flushed to the log before each update.

VITA

HWEI JIUN CHANG

Candidate for the Degree of

Master of Science

Thesis: RECOVERY FOR MEMORY-RESIDENT DATABASE SYSTEMS

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Taipei, Taiwan, November 8, 1958, the daughter of Chin May Shiu and Pei Shang Chang.

Education: Graduated from Ging May Girl Senior High School, Ging May, Taiwan, in June 1977; received Bachelor of Arts Degree in Economics from Chinese Culture University in June 1981; completed requirements for the Master of Science degree at Oklahoma State University in December, 1988.