

Algorithms for the Management of Remote Backup Data Bases for Disaster Recovery

C. Mohan, Kent Treiber

Data Base Technology Institute, IBM Almaden Research Center, San Jose, CA 95120, USA
(mohan, kentt}@almaden.ibm.com

Ron Obermarck¹

IBM, 2800 Sand Hill Road, Menlo Park, CA 94025, USA

Abstract For high availability in disaster recovery situations, it is desirable to reflect data base changes made by a transaction processing system continuously on a (replicated) data base maintained at a remote site. To keep the resource consumption on the backup system low, we describe a general, low overhead method for exploiting parallelism in the processing of the log records received at the backup system. Our approach is general enough to accommodate even the ARIES-type recovery and concurrency control methods which support high concurrency and high efficiency via write-ahead logging, nested transactions, operation logging and semantically-rich modes of locking. We also propose techniques for (1) checkpointing the state of the *backup* system so that recovery can be performed quickly in case the backup system fails, and (2) allowing new transaction activity to begin even as the backup is taking over after a primary failure. We present some performance measurements from a prototype. We discuss distributed transactions, shared disks (*data sharing*) transaction environment and combining executions of *1-Safe* and *2-Safe* transactions in a single system.

1. Introduction

As the pace of computerization increases and the role of electronic information in the success of an organization is enhanced, the importance of protecting such information from disastrous situations like hurricanes, fires, and terrorism become more and more important. Therefore, for high availability in disaster recovery situations, it is desirable to reflect data base changes made by a transaction processing system (call it the *primary system*) on a (replicated) data base at a remote site (the *backup system*). For a long time, there has been a great demand from customers, especially banks and insurance companies, for transaction systems to support such a configuration efficiently. Unfortunately, this problem has gained significant attention only in the last few years from both industrial and academic researchers [BuTr90, GaPo90, KHGP91, Lyon88, Lyon90, PoGa92]. In the meantime, customers have been forced to develop their own ad hoc solutions which do not perform very well since they are implemented outside of the transaction systems.

Since the work done on the backup system is pure overhead in the sense that, until a disaster occurs, all that work is not going towards handling additional workload for the enterprise, we would like to minimize it as much as possible. Resource consumption at the backup should be low relative to the consumption by the primary transaction processing system. Throughput for the processing

on the backup must be better than that achieved by the primary in order to minimize the effects of network and backup system failures on how fast the backup can then catch up with the primary. A concurrency control technique is required to ensure that the backup's data base is transaction consistent and identical (or nearly identical) to that managed by the primary. A technique is also needed for checkpointing the state of the *backup* system so that the latter could be recovered quickly in case it encounters a failure.

One possible strategy for interactions between the primary and the backup systems is that the primary system commits a transaction independent of the backup system. That is, the primary first commits the transaction and later propagates that decision to the backup by sending the commit log record. This is called *1-Safe* execution. With this type of execution, the fact that a transaction had executed *and committed* may not be made known to the backup system, if a failure of the primary were to happen at an inopportune time. That is, a committed transaction may be *lost*. With *2-Safe* execution, the primary uses a two-phase commit protocol (e.g., the current industry-standard Presumed Abort commit protocol of [MBCS92, MoLO86]) with the backup to ensure that the decision to commit is arrived at *jointly* by the two sides. That is, a committed transaction will not be *lost*. Most of the systems and proposals in this area [BuTr90, KHGP91, Lyon90] favor the *1-Safe* execution strategy due to its performance advantages at the risk of losing some transactions. Some of the other tradeoffs between *1-Safe* and *2-Safe* executions are discussed in [GaPo90].

We use a log-based approach for implementing the remote backup support. We are not alone in dismissing the classical distributed data base approach as being inadequate for disaster recovery purposes [GaPH90, GaPo90, KHGP91, Lyon88, Lyon90, PoGa92]. Tandem™ already has a product called Remote Data Facility (RDF™) and IBM has a product called IBM Remote Recovery Data Facility [IBM91]. These systems are almost always based on shipping the log to a remote site and using the shipped log records to keep the remote data base up to date.

The rest of this paper is organized as follows. Our method is described in detail in section 2. We discuss how parallelism is exploited in the processing of the log records in the backup system. We also describe a technique for checkpointing the state of the backup so that the backup can recover quickly after its own failure. Another technique which is described in this section allows new transaction activity to begin even as the backup is taking over

¹ Current Address: DEC, 800 W. El Camino Real, Mountain View, CA 94040

the role of the primary when the old primary fails. A performance study relating to an implementation of our method in the context of IMS is summarized in section 3. Section 4 discusses some of the implications of supporting 1-Safe and 2-Safe executions of transactions in the same system on a per-transaction basis. Section 5 discusses some of the implications of supporting distributed transactions and the shared disks (*data sharing*) transaction environment in the remote backup context. We conclude by presenting a summary in section 6.

2. Our General Method

In this section, we introduce our general approach for solving the problems relating to the support of remote backups. First, we state the assumptions behind our work. Next, we discuss how our general method functions. Our approach exploits parallelism. In performing the tracking of the primary's updates at the backup, our method tries to take advantage of the existing code for performing restart recovery after a failure of the transaction system.

2.1. Assumptions

Some of the assumptions that we make in our general approach are:

- (1) The primary and the backup are connected by high-speed communication links. The communication protocol guarantees message delivery and preserves message ordering.
- (2) The primary follows the 1-safe execution strategy. The implications of mixing 1-safe and 2-safe executions in the same system are discussed in the section "4. Hybrid of 1-Safe and 2-Safe Executions".
- (3) The Write-Ahead Logging (WAL) approach to recovery is used [Gray78, MHLPS92]. In WAL systems, an updated page is written back to the same disk location from which it was read. The *WAL protocol* asserts that the log records representing changes to some data must already be on stable storage *before* the changed data is allowed to replace the previous version of that data on disk. Each log record is assigned, by the log manager, a unique *log sequence number (LSN)* at the time the record is written to the log. The LSNs are assigned in ascending sequence. On finishing the logging of an update to a page, the LSN of the log record corresponding to the *latest* update to the page is placed in a field (*page_LSN*) in the page header. The buffer manager examines the *page_LSN* value to ensure that the log has been forced up to that LSN before it writes a modified page to disk.
- (4) The transaction system supports the *steal* and *no-force* buffer management policies [HaRe83]. This means that pages with uncommitted updates may be written to disk and that at commit time all the pages updated by a

transaction are *not* required to be written to disk, respectively. One consequence of following these policies is that log records written by transactions will contain enough information in order to undo as well as redo the transactions' updates. Typically, each log record representing data base updates refers to the updates performed on a single page of the data base, as in ARIES [MHLPS92]. Depending on the locking protocols used, each log record may describe the update to a page using before and after images of the modified fields of a record within the page, or using an operational description (e.g., add 5 to field 3 of record 10 on page 15). Variations of our methods can support *no-steal* and *force* policies with redo-only logging such as used by IMS Fast Path [MoTO90].

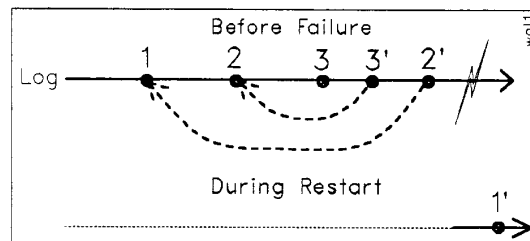
(5) *Compensation log records (CLRs)* are written to describe the updates performed as part of undoing some earlier updates. This means that, if the chaining of CLRs to earlier-written log records were to be performed as in ARIES (see Figure 1), then for those log records whose rollbacks had been completed prior to the failure of the primary (as reflected in the log records received at the backup), no undo will be performed again.

(6) The data base descriptions of those objects that are in both the primary and the backup data bases are the same. In particular, the mapping of objects (e.g., tables) to files is the same. Since the log records only refer to files and not to particular devices where the files reside, the physical mapping of files to devices can be different between the primary and the backup, although the page sizes must be the same.

2.2. Processing During the Tracking Phase

In this subsection, we describe the processing performed by the backup when the primary is alive and the backup is tracking the work done at the primary. Actions taken by the backup system when it is recovering from its own failure and when it takes over the role of the primary are discussed in the following subsections.

Our method takes advantage of the inherent sequencing of log records on a log to support efficient and highly parallel application of changes involving different pages. This processing will result in a backup data base identical to the original data base, as far as the logged changes which



I' is the Compensation Log Record for I. UndoNextLSN chain is shown (1' has a NULL pointer).

Figure 1: ARIES Recovery Scenario - Log Records of a Single Transaction

¹IBM and IMS/ESA are trademarks of International Business Machines Corp. TMF, RDF and Tandem are trademarks of Tandem Computers, Inc. DEC, VAX, VAXcluster and Rdb/VMS are trademarks of Digital Equipment Corp. Oracle is a registered trademark of Oracle Corp. Tuxedo is a registered trademark of Unix System Laboratories, Inc. Transarc is a registered trademark of Transarc Corp. Encina is a trademark of Transarc Corp. X/Open is a trademark of X/Open Company Ltd.

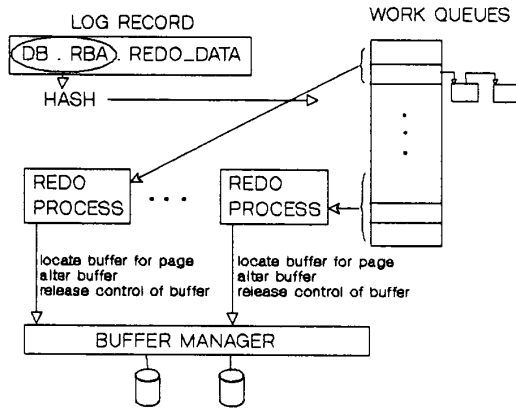


Figure 2: Parallel Redo Processing

had been received by the backup are concerned. With this technique, information that the transaction processing system places on its recovery log is transported to the backup system, either in parallel with writing the log data to a local device or at some later time. At the backup system the log records are placed on stable storage. A single "hash process" also places a the subset of these records required for *repeating history* (redo) on a large number of *work queues* (see Figure 2). Transaction state change log records (e.g., *Begin_Transaction*) and *undo-only* log records are not placed on these queues since they are not needed for redoing updates. So, on a take-over, the *undos* of incomplete transactions will be performed by processing the *stable storage* version of the log in reverse chronological order, as in ARIES and other recovery methods. Note that, even if there are no undo-only log records, the log records in the work queues cannot be used for undo since they are not maintained in reverse chronological order with respect to log records relating to *different* pages. The reverse chronological order processing across different pages also is important in order to ensure that the writing of CLR's and the tracking of the progress of rollback are done properly (as is done using the UndoNxtLSN pointer in the example of Figure 1). The latter ensures that each undoable log record is undone only once (see [MHLPS92, MoPi91, RoMo89]). That type of processing also ensures that completed structure modifications like page split and page delete will not be undone (see [Moha92, MoLe92]).

Many parallel processes can remove records from these work queues and apply the changes to the data base. In the simplest case, a separate work queue exists for each data base page. As each redo record is retrieved from the log shipped by the primary, the log record is placed on the work queue corresponding to the page referenced in the log record. A parallel server process (one for each work queue) can remove one redo record at a time from the work queue and apply its change to the data base. This server process must remove redo records from a work queue in the same sequence as they are placed on the work queue. These actions are illustrated in Figure 2.

In reality, it is not practical to use a different work queue for every page. So, a smaller number of work queues (a few hundred) will be used, with changes for more than one page being placed on each work queue. The primary requirements to be satisfied by this scheme are:

- All redo records associated with a particular page must be placed on the same work queue. One possibility is to associate only one work queue with *all* the pages of the data resident on a particular disk. With this approach, contention for the same disk arm between different server processes can be avoided.
- Redo records must be placed on their associated work queue and subsequently applied to the data base in the same sequence as they are placed on the log. No sequencing needs to be enforced between *different* work queues.

In practice, these requirements generally imply that a work queue has only a single server, but a single server can process work from many queues. Assignment of a work queue to a server can be managed dynamically based on the workload. Assignment of redo records to work queues can be accomplished by hashing the information that identifies the page and using the hash value to specify the work queue, thus spreading the work across the queues.

Buffer management in the backup can be done the same way it is done on the primary [TeGu84]. That is, background writes to disk of modified pages can be batched to reduce CPU costs and to improve I/O efficiency, LRU policy or a variation can be followed for buffer replacement and so on. Because we prefer to avoid doing single-page read or write I/Os and because the numbers of log records associated with the different work queues at any given time may be very different, we use a shared buffer pool across the different work queues. The size of the buffer pool in the backup does not have to be as large as the one in the primary since the pages which were read in the primary but were not modified would not be brought into the buffer pool at the backup. With respect to write-ahead logging, if the log records received from the primary are distributed to the work queues *even before* they are safely recorded on stable storage at the backup site, then the buffer manager on the *backup* also has to enforce the WAL protocol. On the other hand, if the log records are placed on stable storage at the backup site *before* they are placed on the work queues, then there is no need for the buffer manager to enforce the WAL protocol.

The technique described above for exploiting parallelism is very general in that it could also be used in the context of restart and media recovery of the primary system to improve the time it takes to complete such processing. The idea of using parallelism to perform faster *restart* undo and redo processing is discussed further in [MHLPS92]. The kind of processing of the log performed in ARIES during the analysis pass may also be used with our method to identify, for each file on which redo needs to be performed, the set of pages of the file which need to be brought into memory for potentially being updated using the log records. Then, for each file, a single I/O call (or at least a small number of I/Os) may be issued to bring into memory all the required pages of that file. Doing such

batched I/Os, instead of issuing one I/O call for every desired page, will reduce CPU overheads and I/O latencies.

2.3. Checkpoints and Failures of the Backup

In order to handle failures of the *backup* system, the backup system should periodically take its own checkpoints. Note that the checkpoint log records written by the *primary* cannot be used by the backup since the buffering and disk write activities for specific pages on the primary and the backup will be very different. The *backup* checkpoints should keep track of how much redo activity has been performed and externalized to the disk version of the backup data base. The checkpoints should also record the state of the work queues to know how much work had remained unprocessed. Since the primary system may keep producing new log records even when the backup system is down, it is important to make the restart of the backup system be very efficient and fast. The checkpoint scheme described in this section is an enhanced version of the one described in [MHLPS92].

To provide such a fast backup restart capability, the buffer manager at the backup must keep track of the REDO_LSN for each *dirty* page in the buffer pool. The *page REDO_LSN* is the LSN of the *first* log record which dirtied the page after the page was last written to disk or after the page was first read in from disk, whichever happened most recently. It is assumed that one or more background processes periodically sweep through the buffer pool and write to disk dirty pages with relatively old REDO-LSNs, thereby changing those pages' states from *dirty* to *clean*.

Associated with each *work queue* also is a REDO_LSN. This *work queue REDO_LSN* is the LSN of the *oldest, unprocessed* log record in that work queue (i.e., the log record which is not yet applied to the relevant data base page in the buffer pool). This information is needed to repopulate the work queue with log records during restart after a failure of the backup system. If a particular queue is empty, then the REDO_LSN associated with that queue is the LSN of the *next* log record that the "hash process" finds in its input queue.

At checkpoint time, the following are recorded by the backup in a well-known place on stable storage: (1) for each dirty page in the buffer pool, the page number (including the data base ID) and its REDO_LSN; (2) for each work queue, the work queue identifier and its REDO_LSN. Note that there is no need to stop all processing in the backup system when this checkpoint is taken. That is, the backup's checkpoint can also be a *fuzzy* one, like the primary's checkpoint [MHLPS92].

At the time the backup system restarts after its own failure, the information recorded at the time of the most recent checkpoint before the failure is read in and processed. The LSN for the beginning of the reprocessing of the log is chosen to be the *minimum* of the LSNs recorded in the checkpoint. During this log scan, if any log record is encountered whose LSN is *less than* the corresponding work queue's REDO_LSN, then that log record is *not* processed, unless the record's LSN is *greater than or equal to* the corresponding page's REDO_LSN (for pages which did not appear in the checkpoint record, the REDO_LSN is assumed to be infinity). Reprocessing involves putting back the log record in the work queue and handling it as in the

backup system's normal processing. When handling a log record, its update will be redone only if the referenced page's LSN is *less than* the log record's LSN. This is the same condition that ARIES checks during restart redo processing.

Note that since the backup's checkpoint log record contains enough information it is possible to reopen, during recovery after a backup failure, in parallel all the files that would be involved in restart processing. The reopening of the files could be done, using multiple processes, in parallel with the processing of the log which results in the rebuilding of the work queues. Remembering in the checkpoint the per-page and the per-work-queue LSNs helps in avoiding some unnecessary I/Os at backup-restart time. Also, it enables the initiation of batched (i.e., multipage) I/Os for reading in those pages that would need to be accessed during the reprocessing. These are the pages whose IDs appear in the checkpoint log record.

2.4. Processing on Takeover by the Backup

While redo records are being applied to the backup data base, the data base state would be inconsistent. Since no locks will be held which would shield these inconsistencies, data base accesses by normal transactions cannot be supported. For disaster recovery applications, this is not a problem (in reality, customers would like to be able to process *queries* against the backup data base). Consistency is required only at the point that a disaster occurs and recovery is initiated by making the backup take over the processing from the primary. When the backup system is asked (presumably by a human operator) to take over the role of the primary, to obtain a transaction consistent state, the backup system must (1) finish processing the log and then (2) perform standard rollback processing for all uncommitted transactions: apply the undo log records for uncommitted transactions in reverse chronological sequence, as in the undo pass of the ARIES and other recovery methods.

If it is desired to make the takeover occur in such a way that *new* transactions could be processed as soon as possible, then the backup system may be made to keep track of the locks acquired by the in-flight transactions before they perform their updates. With this change, once takeover is initiated and the redo pass is completed, rollbacks of the uncommitted transactions may be performed in parallel with the processing of new transactions since locks would protect the uncommitted updates.

If we are even more ambitious about allowing faster access to data by new transactions once takeover has been *initiated*, then we can extend the method described in [Moha91] to allow that. That method was proposed to permit data accesses by new transactions even before a traditional transaction system completes performing *restart* recovery. That proposal does not involve reacquiring locks or developing too much new code. It is a very simple and cheap solution. That method exploits the information (e.g., IDs of *dirty* pages as in ARIES and IDs of dirty files as in DB2) that is logged at the time of a checkpoint in a transaction system and that is brought up to date as of crash time by the analysis pass of recovery [MHLPS92]. This dirty list lets the system know what pages *may* be modified during the redo pass. As long as the *redo* pass is in progress, new transactions are not al-

lowed to access these pages. In order to determine what pages may undergo modifications during the *undo* pass, the method of [Moha91] makes use of the *Commit_LSN* idea of [Moha90b] in a novel way. Essentially, until recovery is completed, any page whose LSN is *greater than or equal* to the LSN of the oldest update which needs to be rolled back is not allowed to be accessed by new transactions since such a page *might* have some uncommitted updates which are not protected by locks. There are some additional details which are described in [Moha91].

In the current remote backup context, the above method can be easily adapted to allow new transactions to access those pages which will not be modified when the backup system, in the process of taking over the role of the primary, performs its redo and undo processing. In order to do this, the backup, while writing to stable storage the log records received from the primary, needs to keep track of what transactions are active and what the LSN of the oldest uncommitted update is. An alternative would be for the backup to wait until it begins to take over the role of the primary and at that time use the active *transaction* information from the last checkpoint record that it had received from the now-failed primary and bring that information up to date as of the end of the log, as in the analysis pass of ARIES. Note that the dirty *objects* (page or file) information from that checkpoint record cannot be used since that information reflects the state of the buffer pool in the *old primary* which could be very different from that of the backup. But the needed dirty page information for the backup can be easily computed by unioning the list of IDs of the dirty pages in the *backup's* buffer pool and the list of page IDs referenced by the redo log records remaining to be processed in the work queues. Once these pieces of information are collected, new transaction activity can be allowed to begin and the tests described in [Moha91] can be made to ensure that the new transactions access only those pages that will not be modified during the time the new primary finishes processing the log for redo and undo.

2.5. Advantages of Our Method

Below, we list some of the advantages of our parallel processing method for log processing at the backup site.

- (1) Synchronization is not required between update of the actual data base and update of the backup data base. This has availability and performance benefits: update of the actual data base can occur even if the remote site (or part of the remote site) is not available; transfer of log records to the remote site can be batched for efficiency.
- (2) Complex serialization mechanisms like traditional data base locking are not required. Thus, their pathlength and software development costs are avoided.
- (3) Many parallel servers can easily be utilized. Since typically in a DBMS a wait occurs when I/O is required, parallelism is important to get parallel I/Os as well as to utilize more than one processor in a multiprocessor.
- (4) Since, for a given page, all its log records are processed in the same order in which the corresponding updates were applied in the primary system, our method can support, *without any special additional logic*, all the features that ARIES and its extensions [Moha90a, Moha92, MoLe92, MoPi91, RoMo89] support. These features in-

clude support for partial rollbacks, semantically-rich modes of locking, operation logging, nested transactions, fuzzy checkpointing, page-oriented redos, logical undos, fuzzy dumping, media recovery, and so on. Some of these protocols allow the *uncommitted* updates of one transaction on a given page to be moved to another page by different transactions.

(5) Simplicity and ease of implementation: The amount of extra software that needs to be written is much smaller compared to the alternative approaches and much of that software can be written as a separate component.

(6) Performance:

- **Low Pathlength** The pathlength (i.e., number of instructions executed) at the backup would be much smaller than that at the primary since locking and *logical* data management (e.g., traversing index pages and performing read operations of index and data pages) are not required at the backup.
- **High Throughput** Redo processing can operate at several times the rate of normal transaction processing. The bottleneck in throughput will normally be the disk I/O - one of the data base disks will saturate. Because the servers are asynchronous, temporary saturation is not a problem - the queues for that server will simply grow longer.
- **Reduced I/Os** Even though the primary system may force to disk at commit time the pages modified by a transaction (see [MoTO90]), the backup does not have to do that. If the same page were to be modified by a number of transactions in close succession, the primary may write that page many times to disk whereas the backup may write it only once. This *no-force* policy at the backup could reduce the number of I/Os dramatically. Another way to reduce the number of I/Os is by doing bulk I/Os for reading in pages for which redo log records exist. This can be done by analyzing the work queues and identifying a set of pages to be brought in.

(7) Unlike some algorithms [KHGP91], our method does not require that any additional logging be done at the primary system to support the remote backup algorithm.

3. Prototype Work

The Lifeboat project [BuTr90] at the IBM Almaden Research Center has implemented our remote backup algorithm for providing disaster recovery support for IMS/ESA™. Figure 2 illustrates this implementation (more details of this implementation and the adaptation of our general algorithm for IMS are described in [MoTO90]). We have performed a number of experiments to validate correct operation, developed a working load-balancing scheme and measured performance. Due to space constraints, only performance will be discussed.

The experiments that we performed were based on running an IMS/ESA benchmark whose transaction mix had been selected to cause heavy update activity. The performance of the primary system was measured (CPU consumption and log creation rate) and the log data was saved. The log data was then fed to a backup system at various rates to measure performance and find bottlenecks. We are interested in two key measurements:

- **Backup CPU consumption relative to primary CPU consumption** (backup/primary). This certainly will change with different workloads, but we have used a heavy update workload to be conservative.
- **Maximum redo ratio** (backup log processing rate/primary log creation rate). After an outage of the network or the backup system where the primary system continues to perform updates and produce log records, a high tracking rate is necessary to get the backup system data bases current (we call this *catchup*). We feel that a redo ratio of two times the peak primary rate is a reasonable goal. At this rate, currency can be reached in two times the outage time.

In our environment, the first bottleneck encountered was in the I/O area. In particular, the utilization of certain data base disk devices was driven over 90%, driving service times very high and limiting throughput. A worst case backup system will have the same I/O rate per log record as the primary system, so trying to drive the redo ratio above one may require better I/O performance at the backup site than at the primary. Our solution to our bottleneck was to spread the high activity files over more disks. The bottlenecks on the primary system may be very different from the bottlenecks on the backup system due to the fact that read-only accesses to pages on the primary do not cause any activity on the backup. Also, the desire to support a high redo ratio could cause differences.

The second bottleneck is related to the number of redo processes in use. In this particular case, it wasn't a CPU bottleneck, but an I/O bottleneck since a redo process must wait if the buffer manager performs an I/O on its behalf. The solution was to increase the number of redo processes. In practice, our recommendation is that the maximum number of redo processes be twice the peak number of transaction processes at the primary. This seems to give the backup system capacity for a redo ratio well over two.

We removed bottlenecks to the point where we could achieve a redo ratio of almost four. The results of several measurements are shown in Figure 3. Since the details of CPU consumption have not been analyzed, there may be opportunities for improvement in that area. Even without

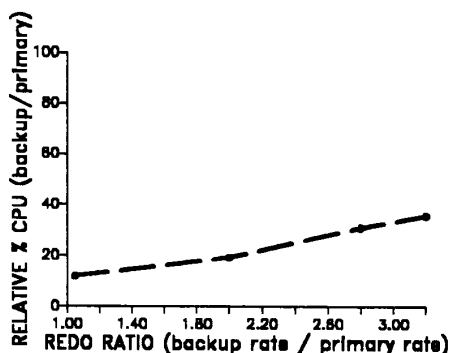


Figure 3: Parallel Redo Experiment Results

improvement, it appears that we have an approach that can keep backup data bases current, achieve very high redo ratios, and do it for a reasonable fraction of the primary transaction system's CPU consumption. CPU consumption appears to be about 1/3 of the consumption described in [PoGa92] but lack of workload specifications makes comparisons difficult.

While our environment did not require CPU parallelism for redo processing, it is easy to describe an environment that does require it: catching up after a network outage where two primary systems are sharing data bases, each consuming 90% of a 6-processor machine. The backup system is running on an equivalent 6-processor machine. Since a redo ratio of two appears to require about 20% of the primary system's CPU, we need $.2 \times 9 \times 6 \times 2 = 2.16$ processors worth of CPU.

4. Hybrid of 1-Safe and 2-Safe Executions

It has been suggested [GaPo90, Lyon88] that to trade-off the additional cost of a 2-Safe execution with the extra guarantee that it provides, a system may implement a hybrid scheme in which, depending on the *value* or importance of a given transaction, the system may choose to execute the transaction 2-Safe or 1-Safe.

The problem with a hybrid scheme like this is that if the backup were to be redoing the primary's updates based on the log records sent by the primary, as we have described before, then we have to be careful if a given transaction (T1) is executed 1-Safe and *later* another transaction (T2) is executed 2-Safe. If T2 depends on T1 and hence T2 needs to follow T1 in the serialization order, then we could have inconsistencies in the data if the 2-Safe transactions' log records are sent right away to the backup while the 1-Safe transactions' log records are sent slowly. If this approach is used, then it is possible that the log records of T2 are received by the backup but not the log records of T1. This could happen if a failure were to bring down the primary before T1's log records are sent. This could destroy data consistency and also cause structural problems in objects like indexes, if an index method like ARIES/IM [MoLe92] were to be used. T2 might have done some key inserts on a page of the index, where the affected page was the one that was allocated due to a page split performed by T1. Our approach to deal with this problem is to make sure, when a 2-Safe transaction is executed and its log records are sent to the backup, that the log records are not sent out of sequence. That is, a side effect of a 2-Safe transaction execution in a hybrid environment is to ensure that all log data created *prior* to the 2-Safe transaction commit is on stable storage at the backup site *before* the 2-Safe transaction is allowed to commit. This will include 1-Safe transactions' changes. In the above example, since T2 used a 2-Safe execution, T2 will commit at the primary only after all the log records preceding and including T2's last log record had been received by the backup and stored on stable storage. As a result, all log records of T1 (which must appear before T2's last log record) would have been received by the backup.

5. The Shared Disks Environment and Distributed Transactions

There are additional complexities to deal with whenever more than one log contains information about a single transaction [MoLO86] or a single recoverable resource [MoNa91]. This then includes transactions executing in distributed and shared disks (SD) (also called *data sharing*) environments, respectively. With SD, all the disks containing the data base are shared among the different systems in the complex. Every system that has an instance of the transaction system executing on it may access and modify any portion of the data base on the shared disks. Since each transaction system instance has its own buffer pool and because conflicting accesses to the same data may be made simultaneously from different systems, the interactions among the systems must be controlled by the use of global locking facilities and protocols for the maintenance of the coherency of the data buffered (*cached*) in the different systems.

Just as data sharing or distributed transactions add additional protocols to deal with recovery, so must remote site recovery that supports them. Commercially, SD is a very popular approach. SD is the approach used in IBM's IMS/VS Data Sharing product [StUW82], TPF product [Scru87] and the Amoeba research project [MoNa91, MoNa92a, MoNa92b, MoNS91, SNOP85], and in DEC's VAX Rdb/VMS™ [ReSW89]. More recently, for the VAXcluster™ environment, third-party DBMSs like ORACLE™ and INGRES have been modified to support SD. Hitachi and Fujitsu also have products which support the SD environment.

The Presumed Abort commit protocol [MBCS92, MoLO86] was invented and implemented in the context of R*. It is a highly optimized variant of the classical two-phase commit protocol [SBCM93]. It has since then been implemented in IBM's QuickSilver, CMU's Camelot, Tandem's TMF™, Transarc's Encina™, USL's Tuxedo™ and DEC's VMS. It is being adopted as part of the ISO and X/Open™ distributed transaction processing standards. Given the popularity of this protocol, it is important that we analyze its implications in the context of remote backups. If the DBMS were to support distributed transactions and the Presumed Abort protocol were to be used, then we have to be careful since, once the coordinator receives acknowledgements (for the commit messages) from all its subordinates, it erases from virtual storage the information which says that the transaction had committed. If a subordinate site's system were to go down and its backup were to take over, then it is possible for the backup to have received only the *Prepare* log record and not the *Commit* log record, when in fact the primary had logged the commit and acknowledged to its coordinator. Under these conditions, if the backup were to query the coordinator, then the backup may be given the presumed decision of abort (rollback), which would lead to inconsistencies between the coordinator's and subordinate's data bases. This could happen because by the time the backup subordinate makes its enquiry the coordinator might have received all the required acknowledgements for the commit messages and then forgotten the transaction. One way to deal with this problem is to make the coordinator (1) be aware that the inquiry is coming from the *backup* of a subordinate and (2) not give the presumed decision. If infor-

mation about the actual decision is still available (e.g., in virtual storage or by accessing the log), then that information can be passed back. Another option is to make a subordinate delay sending its acknowledgement until it ensures that its backup has received the commit log record and stored it on the backup's stable storage. This delay should not affect lock-hold times, etc. since a subordinate could release the locks held by the committing transaction *before* propagating the commit record to its backup.

When the primary system is using SD, then inconsistencies may be caused if the backup system is also configured to be an SD with each data sharing backup system being the backup for the corresponding primary system. Inconsistencies may arise since we may no longer be redoing updates to the same page in the same order in which they occurred in the primary system. It would be too expensive to try to make it work by using global locks, etc. Our solution for this case is to (1) use a single backup system to receive and merge logs from all primary systems that share a given set of disks, (2) use the same backup system to perform redo work, and (3) if the redo workload exceeds the capacity of a single system, route merged log data to redo server systems, partitioning redo processing to prevent conflicts between servers. Note that with this approach, even if one of the data sharing primary systems encounters a disaster and it is not sufficient to keep the primary going with the remaining data sharing system, then the surviving primary must be brought down before letting the backup system take over.

6. Summary

We presented a method for managing a remote backup data base for the purpose of providing protection from disasters which destroy the primary data base. Our method is general enough to accommodate the ARIES-type recovery and concurrency control methods as well as the methods used by other systems such as DB2, DL/I and IMS Fast Path. It provides high performance by exploiting parallelism and by reducing I/Os via different means like log analysis and choosing a different buffer management policy from the primary.

The techniques described here are very general in that they could also be used in the context of restart and media recovery of the primary system to improve the time it takes to complete such processing. We proposed a technique for checkpointing the state of the backup system so that recovery can be performed quickly, in case the backup system were to fail. The checkpointed state includes the states of the in-memory work queues and the dirty pages in the buffer pools. We also presented a technique which allows new transactions to begin page accesses for read and *update* even before the takeover processing is completed by the backup.

We presented some performance results relating to an implementation of our method in the context of IMS. We discussed some problems relating to distributed transactions, the shared disks environment, and combining executions of *1-Safe* and *2-Safe* transactions in a single system. We proposed some possible solutions for dealing with these problems.

7. References

- BuTr90** Burkes, D., Treiber, K. *Design Approaches for Real-Time Transaction Processing Remote Site Recovery*, Proc. IEEE Compcon Spring '90, March 1990.
- GaPH90** Garcia-Molina, H., Polyzois, C., Hagmann, R. *Two Epoch Algorithms for Disaster Recovery*, Proc. 16th International Conference on Very Large Data Bases, Brisbane, August 1990.
- GaPo90** Garcia-Molina, H., Polyzois, C. *Issues in Disaster Recovery*, Proc. IEEE Compcon Spring '90, March 1990.
- Gray78** Gray, J. *Notes on Data Base Operating Systems*, In *Operating Systems - An Advanced Course*, R. Bayer, R. Graham, and G. Seegmuller (Eds.), Lecture Notes in Computer Science, Vol. 60, Springer-Verlag, 1978.
- HaRe83** Haerder, T., Reuter, A. *Principles of Transaction Oriented Database Recovery - A Taxonomy*, Computing Surveys, Vol. 15, No. 4, December 1983.
- IBM91** *IBM Remote Recovery Data Facility, Announcement Letter 291-500*, IBM, September 1991.
- KHGP91** King, R., Hallm, N., Garcia-Molina, H., Polyzois, C. *Management of a Remote Backup Copy for Disaster Recovery*, ACM Transactions on Database Systems, Vol. 16, No. 2, June 1991, P338-68.
- Lyon88** Lyon, J. *Design Considerations in Replicated Data base Systems for Disaster Protection*, Proc. IEEE Compcon Spring '88, March 1988.
- Lyon89** Lyon, J. *Transaction Commit issues in RDF2*, Proc. 3rd International Workshop on High Performance Transaction Systems, Asilomar, September 1989.
- Lyon90** Lyon, J. *Tandem's Remote Data Facility*, Proc. IEEE Compcon Spring '90, March 1990.
- MBCS92** Mohan, C., Britton, K., Citron, A., Samaras, G. *Generalized Presumed Abort: Marrying Presumed Abort and SNA's LU 6.2 Commit Protocols*, IBM Research Report RJ8684, IBM Almaden Research Center, March 1992.
- MHLPS92** Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P. *ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging*, ACM Transactions on Database Systems, Vol. 17, No. 1, March 1992.
- Moha90a** Mohan, C. *ARIES/KVL: A Key-Value Locking Method for Concurrency Control of Multiaction Transactions Operating on B-Tree Indexes*, Proc. 16th International Conference on Very Large Data Bases, Brisbane, August 1990. A different version of this paper is available as IBM Research Report RJ7008, IBM Almaden Research Center, September 1989.
- Moha90b** Mohan, C. *Commit LSN: A Novel and Simple Method for Reducing Locking and Latching in Transaction Processing Systems*, Proc. 16th International Conference on Very Large Data Bases, Brisbane, August 1990.
- Moha91** Mohan, C. *A Cost-Effective Method for Providing Improved Data Availability During DBMS Restart Recovery After a Failure*, Proc. 4th International Workshop on High Performance Transaction Systems, Asilomar, September 1991. Also available as IBM Research Report RJ8114, IBM Almaden Research Center, May 1991.
- Moha92** Mohan, C. *ARIES/LHS: A Concurrency Control and Recovery Method Using Write-Ahead Logging for Linear Hashing with Separators*, To appear in Proc. 9th International Conference on Data Engineering, Vienna, April 1993. A longer version is available as IBM Research Report RJ8682, IBM Almaden Research Center, March 1992.
- MoLe92** Mohan, C., Levine, F. *ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging*, Proc. ACM SIGMOD International Conference on Management of Data, San Diego, June 1992. A longer version of this paper is available as IBM Research Report RJ6846, IBM Almaden Research Center, August 1989.
- MoLO86** Mohan, C., Lindsay, B., Obermarck, R. *Transaction Management in the R* Distributed Data Base Management System*, ACM Transactions on Database Systems, Vol. 11, No. 4, December 1986.
- MoNa91** Mohan, C., Narang, I. *Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment*, Proc. 17th International Conference on Very Large Data Bases, Barcelona, September 1991. A longer version is available as IBM Research Report RJ8017, IBM Almaden Research Center, March 1991.
- MoNa92a** Mohan, C., Narang, I. *Efficient Locking and Caching of Data in the Multisystem Shared Disks Transaction Environment*, Proc. International Conference on Extending Data Base Technology, Vienna, March 1992.
- MoNa92b** Mohan, C., Narang, I. *Data Base Recovery in Shared Disks and Client-Server Architectures*, Proc. 12th International Conference on Distributed Computing Systems, Yokohama, June 1992.
- MoNS91** Mohan, C., Narang, I., Silen, S. *Solutions to Hot Spot Problems in a Shared Disks Transaction Environment*, Proc. 4th International Workshop on High Performance Transaction Systems, Asilomar, September 1991. Also available as IBM Research Report RJ8281, IBM Almaden Research Center, August 1991.
- MoPi91** Mohan, C., Pirahesh, H. *ARIES-RRH: Restricted Repeating of History in the ARIES Transaction Recovery Method*, Proc. 7th International Conference on Data Engineering, Kobe, April 1991.
- MoTO90** Mohan, C., Treiber, K., Obermarck, R. *Algorithms for the Management of Remote Backup Data Bases for Disaster Recovery*, IBM Research Report RJ7885, IBM Almaden Research Center, December 1990; Revised June 1991.
- PoGa92** Polyzois, C., Garcia-Molina, H. *Evaluation of Remote Backup Algorithms for Transaction Processing Systems*, Proc. ACM SIGMOD International Conference on Management of Data, San Diego, June 1992.
- ReSW89** Rengarajan, T.K., Spiro, P., Wright, W. *High Availability Mechanisms of VAX DBMS Software*, Digital Technical Journal, No. 8, February 1989.
- RoMo89** Rothermel, K., Mohan, C. *ARIES/NT: A Recovery Method Based on Write-Ahead Logging for Nested Transactions*, Proc. 15th International Conference on Very Large Data Bases, Amsterdam, August 1989. A longer version of this paper is available as IBM Research Report RJ6650, IBM Almaden Research Center, January 1989.
- SBCM93** Samaras, G., Britton, K., Citron, A., Mohan, C. *Two Phase Commit (2PC) Optimizations and Trade-Offs in the Commercial Environment*, To appear in Proc. 9th International Conference on Data Engineering, Vienna, April 1993.
- Scru87** Scrutchin, T. *TPF: Performance, Capacity, Availability*, Proc. IEEE Compcon Spring '87, San Francisco, February 1987.
- SNOP85** Shoens, K., Narang, I., Obermarck, R., Palmer, J., Silen, S., Traiger, I., Treiber, K. *Amoeba Project*, Proc. IEEE Compcon Spring '85, San Francisco, February 1985.
- StUW82** Strickland, J., Uhrowicz, P., Watts, V. *IMS/VS: An Evolving System*, IBM Systems Journal, Vol. 21, No. 4, 1982.
- TeGu84** Teng, J., Gumaer, R. *Managing IBM Database 2 Buffers to Maximize Performance*, IBM Systems Journal, Vol. 23, No. 2, 1984.