# Performance of Recovery Algorithms for Centralized Database Management Systems

Vijay Kumar and Shawn D. Moe
Computer Science Telecommunications
5100 Rockhill
University of Missouri-Kansas City
Kansas City, MO 64110, U.S.A.
kumar@cstp.umkc.edu

**Abstract**: Database Recovery is responsible for preserving the database consistency after a failure of any kind (transaction, system or media). Relevant information solely for recovery is saved in a log during normal transaction processing. To recover from a failure, basically two operations: *undo* and *redo* are applied with the help of the log on the last consistent state of the database. These two operations can be combined in four different ways to define four different types of recovery algorithms: "undo-redo", "no undo-redo", "undo-no redo" and "no undo-no redo". Each of these algorithms manages log and updates to the database differently, which affect the overall performance and the availability of the database.

To our knowledge, not much work has been done on the performance of recovery algorithms. There are only six reports available and these works have concentrated their studies only on a few algorithms. They have mainly used queuing approach, which we believe is not adequate for a detailed performance study. In this paper we investigate the relative performance of these four algorithms through simulation. The purpose of conducting this study is first to verify the findings of earlier works, and second to obtain a detailed comparison of the behavior of all recovery algorithms. Earlier works . Our simulation studies show that in most cases, undo-redo and no undo-redo deliver similar recovery performance but better than undo-no redo and no undo-no redo. We observe that the recovery times increase with *multiprogramming level*, but generally decrease as the number of *checkpoints* increases. The undo-no redo algorithm results in a greater number of transaction rollbacks, which creates a larger transaction log, and thus longer recovery times. Its performance increases against undo-redo and no undo-redo as input/output performance improves. Slow input/output performance is also attributed as the cause of the no undo-no redo algorithm's poor recovery performance.

## 1. Introduction

A database may get corrupted either by a transaction failure, by a system failure, or by a media failure. The effect of a transaction failure is usually localized to its execution domain (the number of data items it accessed so far). Consequently, the portion of the database needing recovery is identifiable. A system or a media failure may or may not corrupt any portion of the database, but if it does then the corrupted portion is not identifiable. For this reason, one generally assumes that entire database may be inconsistent and, therefore, must be recovered.

A database recovery is performed with two main operations: *Undo* and *Redo*. An undo operation removes the effects of failed transaction from the database and a redo operation preserves the *ACID* properties of transactions, i.e., a redo completes the left over portion of a transaction commit. Undo and Redo operations can be combined in four different ways to define four different basic recovery algorithms. In this paper we will call them as: U-R (Undo-Redo), NU-R (No Undo-Redo), U-NR (Undo-No Redo), and NU-NR (No Undo-No Redo) [BER87].

This paper presents a performance study of these four algorithms. Our justification in conducting this investigation is that previous studies on recovery provide very little information about the behavior of only a small subset of recovery algorithms. In [GEL78] a mathematical model of a transaction-oriented system has been used to study the effect of checkpointing on recovery. [GRI85] presents a mathematical model which compares several recovery protocols, but uses overall transaction throughput, as opposed to the length of recovery as its method of comparison. [KEN85] present recovery performance for special purpose dedicated algorithms. In [REU84] it was reported that a version of the U-R outperformed the U-NR algorithm. In [ANT92] it is shown that a version of the U-NR algorithm outperformed the U-R algorithm. In summary there is no single work that presents a detail performance study of these four algorithms and compare their behavior under a varieties of failure environments. Our aim in studying the performance of these four different types of algorithms is two fold. First, we want to identify the most favorable algorithm, i.e., the algorithm that offers the best performance and second, to verify the findings of earlier works. For this reason we have measured the performance of these algorithms in various failure scenarios. We have investigated the effect of redos, undos, log management and checkpointing frequency on recovery time (time to complete the entire recovery process). All earlier works have used queuing model to investigate the behavior of a few algorithms. We have used simulation modeling to investigate their behavior in detail since it is not easy to do so using queuing model.

The organization of the paper is as follows. Section 2 presents the recovery algorithms we have investigated in pseudo code. Section 3 explains the purpose of checkpointing and Section 4 describes the expected behavior of these algorithms. Section 5 presents the simulation models and parameters used to drive these simulators. Section 6 discusses our findings, and finally Section 7 concludes the paper.

## 2. Recovery Algorithms

In this section we present the recovery algorithms we have investigated. We use TM-DM (Transaction Manager-Data Manager) architecture model [BER87] of database system to describe the working of these algorithms. The TM is responsible for managing concurrent transactions and the DM is responsible for implementing the effects of transactions on to the database. The Recovery Manager (RM) which is a part of DM, uses various schemes to restore the consistent state of the database after a system failure. It determines whether values from uncommitted transactions can update the database, when to

declare a transaction committed, and when transaction modifications are transferred to the stable database. These decisions are made during *forward* transaction processing, and have a direct effect on the type of processing that must be performed during database recovery.

We identify four states of a transaction; *active* , *abort, finished* and *committed.* If a transaction is still executing, then it is *active,* and remains in an *active list*. A transaction unable to remain active because of some error goes to an abort state and enters an *abort list*. A transaction which has completed all its modifications, and has logged its execution history but has not yet installed all its updates into the database is in a finished state. A transaction which has completed all its modifications and has installed them into the database is ready to be *committed.* These transactions wait in a *commit list* to be committed.

The DM of a database system interacts frequently with the underlying operating system. We do not describe the nature of these interaction here, interested readers should refer to [BER87] for detail. So far as the work reported here is concerned, we have made a number of assumptions related to the use of cache for managing data manipulation. To introduce the algorithms in a simple to understand way, we have presented them in pseudo code.

## 2. 1 The U-R Algorithm

The U-R algorithm uses both undo and redo for recovering the database from a failure. An undo is required since the system allows intermediate modifications of a transaction to be installed in the database, and a redo is required because all modifications by committed transactions may not be present in the stable database. We identify the initial value of a data item as BFIM (BeFore IMage) and its new value as AFIM (AFter IMage). A *fetch* operation transfers data from disk to cache while a *flush* moves data from memory to disk.

```
If a transaction T_i wants to write a new value into data item x then
begin
    if T_i is not in the active list then add T_i there;
    if x is not in the cache then fetch it;
    write BFIM (= x) and AFIM (= v) of x to the log;
    overwrite the contents of x by v in the cache;
end
else  if T_i wants to read a data item x then
      begin
          if x is not in the cache then fetch it;
          return the value of x to the transaction
      end
      else  if T_i wants to commit then
```

```
begin
  Add Ti to the commit list;
    {if the system fails before Ti is appended to the commit list, Ti is not committed,
    regardless of the status of the Ti's modifications to the stable database, Ti must be
    undone. Similarly, if a failure occurs after Ti is added to the commit list but before its
    modifications have been transferred to stable storage, Ti must be redone.}
  acknowledge the commitment of Ti to the system;
  delete Ti from the active list;
end
else if Ti is to be aborted then {this is an undo}
    begin
      for each data item x updated by Ti copy the BFIM of x into a cache slot;
      add Ti to the abort list;
      acknowledge the abortion of Ti to the system and delete Ti from the active list;
    end
    else  if there is a system failure then {start recovery}
    begin
      discard all cache slots and fetch log;
      start backward processing of log;
      if data item x is not in cache then get a cache slot for it;
      fetch the commit list;
      if Ti is in the commit list then copy AFIM of x to a cache slot;    {begin redo}
      if Ti is in the abort list then copy its BFIM of x to a cache slot;    {begin undo}
      repeat the process until log records up to the last checkpoint have been processed;
        {this completes the undo and redo operations}
      for each Ti in the commit list, if Ti is in the active list, remove it from there;
    end;
```

## 2.2 The U-NR Algorithm

This class of algorithms requires only undos. To accomplish this all of a transaction's updates are forced in the stable database before that transaction commits. The read, write and abort operations are identical to the U-R case. We present the commit and restore operations here.

```
if a transaction Ti wants to commit then
begin
    if data item x modified by Ti is in the cache then force a flush;
```

{this guarantees that an undo will be performed during recovery}
Add $T_i$ to the commit list; {here all its modifications have been flushed to the database}
acknowledge the commitment of $T_i$ to the system and delete $T_i$ from the active list;
end
else if there is a system failure then
begin
discard all cache slots, fetch log and start backward log processing; {begin undo}
if $T_i$ is in the commit list then skip this log record;
if $T_i$ is not in the commit list then
begin
copy BFIM of x to a cache slot;
repeat the process until all log records up to the last checkpoint have been processed;
remove $T_i$ from the active list;
end;
end;

## 2.3 The NU-R Algorithm

This class of algorithms requires only redos. To achieve this, RM *pins* and *unpins* data in the cache for managing fetch and flush.

if transaction $T_i$ wants to writes the new value v into data item x then
begin
add $T_i$ to the active list if it is not there and if x is not in the cache then fetch it;
write BFIM (= x) and AFIM (= v) to the log;
overwrite the contents of x by v in the cache;
pin the cache slot so that it cannot be flushed and acknowledge write to the system;
end
else if $T_i$ wants to read x then
begin
if x is not in the cache then fetch it and return the value of x to the transaction;
end
else if $T_i$ is ready to commit then
begin
Add $T_i$ to the commit list;
for each data item x modified by $T_i$ unpin the cache slot for x;
acknowledge the commitment of $T_i$ to the system and delete $T_i$ from the active list;

```
end
else if Ti fails then {abort Ti}
    begin
        for each data item x updated by Ti copy the BFIM of x into x's cache slot;
        unpin the cache slot for x and add Ti to the abort list;
         acknowledge the abortion of Ti to the system and delete Ti from the active list;
    end
    else if the system fails then    {begin recovery}
        begin
            discard all cache slots, fetch log and start backward log processing;
            copy AFIM of x to a cache slot if it is not there;
            repeat the process until all log records up to the last checkpoint are processed;
        end;
```

## 2.4 The NU-NR Algorithm

This class of algorithms requires neither undos or redos for recovering the database. To avoid undo, no modifications from a transaction is forced to the stable database before it commits and to avoid redo, all modifications from a transaction must be in the stable database before it commits. This is achieved by shadowing scheme. The location of each data item's last committed value is recorded in a directory maintained on stable storage. When a transaction $T_i$ writes a data item x, a new version of x is created in stable storage. The scratch directory that defines the database state used by $T_i$ is updated to point to this version. When $T_i$ commits, the directory that defines the committed database state is updated to point to the versions that $T_i$ wrote. $T_i$'s modifications are added to the stable database, thus committing $T_i$. Together, these directories point to all of the BFIMs and AFIMs on stable storage thus a transaction execution log is not necessary.

In describing these algorithms, we indicated that data in cache slots are pinned until commit/abort of the transaction that updated the item. Writing updates to database may be implemented without pinning cache slots or in some other way. For example, in some implementations pinning is avoided by using versioning in stable storage. As mentioned before, we did not use a particular implementation technique and just modeled the operation which is right conceptually. The performance of any algorithms not only of recovery, may vary with implementation strategies and we avoided comparing the performance of implementation strategies.

# 3. Checkpointing

Checkpointing is an activity that writes information to stable storage during forward processing to reduce the number of log records that recovery operation must examine. Although checkpointing does

reduce the amount of log processing, it impedes normal transaction processing since the database remains unavailable to new transactions during this process. In some commercial systems, e.g., [ORA] checkpointing and normal transaction processing go concurrently. We have not modeled this aspect since we wanted to study the effect of checkpointing on recovery, and analyzing fuzzy checkpoint during recovery is an expensive process. RM decides at what intervals to perform checkpointing. The interval may be measured in time or in terms of the number of committed transactions since the last checkpoint.

## 4. Expected Behavior of Recovery Algorithms

In this section we speculate the performance of these algorithms, which are then verified by simulation. We expect U-R to provide the best overall forward processing of transactions, while delivering a slower recovery. One of the main reasons for this we believe is the lack of control of RM over cache. RM does not direct or force CM to flush the cache. This lack of control allows CM to implement the most efficient scheme to swap cache pages. The penalty for this freedom appears during recovery, where both undo and redo operations must be performed, however, we expect that a frequent checkpointing may improve its performance.

It is expected that the NU-R algorithm will deliver the best overall recovery performance. The NU-R algorithm requires only redos. At the time of a system failure, under normal circumstances, a large number of transactions are likely to be active, and a much smaller number of transactions would be on the commit list. With this in mind, algorithms requiring Undo (U-R, U-NR) should have more transactions requiring undo. The NU-R algorithm, on the other hand, only has to process the smaller set of transactions residing on the commit list.

The recovery times required for performing the actual undo and redo operations for each data item are the same since the work required is so similar. Since NU-R should have fewer transactions to process, the NU-R algorithm should reflect the best recovery time of the log based algorithms.

It is expected that with each additional checkpoint the overall recovery time will reduce since each checkpoint reduces the number of log records that must be examined during recovery. It is also expected that the recovery time will increases with MPL. With each higher MPL value there should be more transactions (active, to be aborted and to be committed) at the time of failure which will increase the recovery time.

The flexibility of the U-R and NU-R algorithms allow a transaction to become committed faster than with the U-NR algorithm because there is no need to wait until the cache is flushed before the data items are unlocked. Locked data items result in CCM conflicts, which lead to rolled back transactions and additional records written to the log. The logs produced by the NU-R algorithm should be very similar in size to those of U-R, because both algorithms allow transactions to commit before waiting for the cache to

be flushed. It is assumed that NU-R logs may be slightly larger than U-R because a few more data item conflicts are possible due to the delay necessary to unpin the data items during the commit process.

An algorithm that causes more transactions to be aborted should result in a longer recovery time since aborted transactions create additional entries to be written to the log, which may result in a longer recovery time because the additional records must be transferred to memory and then examined. For this reason, the flexibility of U-R should produce the smallest number of aborted transactions. Since U-NR keeps data items locked for a longer period of time, should result in the highest number of aborted transactions. The NU-R algorithm should abort a few more transactions than U-R. This is because NU-R is delayed slightly during the commit process in order to unpin the transaction's data items. This delay is minimal and should not result in a significant difference from U-R.

It is expected that decreasing the I/O transfer time shall also decrease the recovery times in all three algorithms. With better I/O performance, the transaction logs should be transferred to memory faster, resulting in better recovery performance.

The NU-NR algorithm saves its master directory on disk, and so must transfer it and any scratch directories to memory, and then create a new shadow directory. The size of the master directory is likely to be equal to the number of items in the database. With a slow I/O, the directory transfer may take a long time. It is expected that with faster I/O the overall recovery performance of all four algorithms would improve. The relationships between the algorithms' performance, however, should not change dramatically. With the NU-NR algorithm, the directory sizes that must be transferred are constant in size, and thus would not show any relative improvement. The log-based algorithms, on the other hand, will show an improvement in their relative performance.

The NU-NR algorithm seems to suffer performance and resource degradation during forward processing. Several factors contribute to this situation. Access to the stable database is through the directory structure maintained by RM, and thus provides another level of indirection to access data items. The scratch directories, created for each transaction to contain uncommitted versions for each data item, require some management in order to reclaim the space. The most important consideration with this algorithm is the constant fragmentation of the stable database. Since multiple copies of each data item appear on stable storage, it is unlikely that all versions are stored in proximity to each other. Thus a new stable version of a data item may appear in a much different physical location on stable storage, and may even appear on a different physical storage device from where it was previously stored. A database originally created to occupy contiguous storage for I/O efficiency will eventually fragment all over the available disk storage.

With the above factors in mind, it can be argued that the performance of a NU-NR will slowly degrade over the life of the database. We believe that NU-NR would perform better in situations where the percentage of query transactions far outnumbers the update transactions.

U-NR and NU-R algorithms seem very similar since RM exerts control over the timing of CM's activities. During the forward processing, CM is instructed when to flush each modified data item, and when to unpin cache slots. The action of unpinning cache slots should require less work than flushing; unpinning requires only CPU resources, while flushing requires CPU and I/O resources. If the cache is of sufficient size to eliminate massive contention arising from a large number of slots which are pinned and thus cannot be swapped, the NU-R algorithm can be argued to provide better forward processing performance.

The same relationship should exist during a restart as well. In a normal situation, the number of transactions requiring undo should be larger than the number of transactions requiring redo. This situation is assumed because prior to a system failure, the number of active and aborted transactions could outnumber the number of committed transactions, especially at a higher concurrency. The action of undoing or redoing is logically the same. The difference in restart time must then be found in the number of data items to be undone or redone. With this in mind we expect that NU-R would perform the restart faster than U-NR.

With checkpointing, U-NR and NU-R could recover faster than U-R. Logically, restart for either U-NR or NU-R should take less than half the time required for U-R, because (at most) only half the work is required. It is assumed that if the frequency of failure is high, the recovery must be very fast in order to compensate for the unavailability of the database to the users.

## 5. Simulation Model and Parameters

In this section we introduce modeling assumptions and parameters. We have tried to keep our assumptions and parameters consistent with earlier studies to be able to do a meaningful comparison.

It is assumed that the database is a set of data items each of which is a lockable unit. Transaction size (number of data items accessed by a transaction) is exponentially distributed around a mean and the selection of data items is uniformly distributed. The model depicts a closed system where predetermined number of transactions are created and processed after some predefined delay. If a transaction aborts, the same transaction is submitted for processing after a delay of an average response time. We have used WAL (Write Ahead Logging) [BER87] for managing the log. The model uses a strict two-phase locking policy for managing concurrent transactions.

### 5.1. Failure Criteria

To investigate the effect of database failure on recovery, it is simulated using the following criteria.

ActiveBased: failure occurs once the number of active transactions is greater or equal to the supplied threshold value.

CommitBased: failure occurs once the number of committed transactions is greater or equal to the supplied threshold value.

TimeBased: failure occurs once the elapsed simulation time surpasses the threshold value.

TransBased: failure occurs once the number of transactions launched surpasses the threshold value.

CPFullMPL: failure occurs after the first checkpoint is completed and the number of active transactions is equal to the MPL. This criteria was established to always ensure a fully loaded system at the time of failure.

The criteria to flush the log for (U-R, NU-R and U-NR) is also determined by ActiveBased, CommitBased, TimeBased and TransBased as described above. The time to checkpoint the log is also determined using these same criteria. For added flexibility, the models contain a flag that identifies whether transaction log flushing criteria is described for the model. If the flag indicates no criteria, the log is flushed immediately. Otherwise, the criteria is checked to see whether or not to flush the log. In addition to the log flush flag, the model allows various types of checkpoint criteria. This criteria is used to determine what values determine whether the checkpoint threshold has been met. The models for the log-based algorithms assume that a *commit consistent checkpointing* scheme is used to perform checkpointing at the appropriate time.

## 5.2. Model Parameters

In this section we present simulation parameters for the models and explain their meaning.

DatabaseSize: describes the number of data items in the logical database.

NumTrans: defines the number of transactions to be created in the transaction set.

NumItems: defines the average transaction size.

WriteReadPct: defines the percentage of the transactions are to be Write (modification) transactions. The remainder of the transaction set are Read (Query) transactions.

MinimumTransactionSize: defines the minimum number of data items a transaction accesses.

The transaction log is implemented as a sequential list of log records. Each log record contains the owning transaction's identifier, data item identifier, location of the log record (memory or disk), type of log record (read, write, commit, etc.), BFIM, AFIM, and pointers to link the log records for one transaction together.

## 5. 3. Common Forward Processing of Transactions

Since the forward processing of transactions under a recovery scheme shares considerable functionality, it is useful to follow a common transaction scheme. We first explain the common features and then identifies specifics of forward processing for each recovery schemes.

The main simulation process adds transactions to the active list, each after waiting a length of time, which is set to a random derivate drawn from an exponential distribution with a mean of the ThinkTime

(see Section 6.) parameter. The main process is also responsible for determining if the system failure, log flushing, or checkpoint criteria have been met, and then initiating those actions.

The processing of an active transaction includes obtaining the lock on the data item and its manipulation. For simulation purpose, some delays were introduced in the data manipulation in order to keep a transaction active for a period of time. This is necessary to eventually capture active transactions at the time of system failure. In U-R, U-NR and NU-R, the log record is created to reflect the data item modification. After all data items are acquired and modified, the transaction prepares to commit. The commit process is unique for each algorithm, and will be described with the algorithm specific information.

Transaction rollback is performed by releasing data items and updating the lock table. The release of each data item expends simulation time which simulates the Cache Manager (CM) processing to determine if the slot needs to be flushed to stable storage. To simulate a realistic CM, a random number of the held items are determined, which require flushing to stable storage and thus expend additional simulation time. An abort record is written to the transaction log after all of the items have been processed. The transaction is removed from the active list and added to the abort list. The aborted transaction is delayed a random length of time, and then it joins the set of transactions available for processing at the head of the list. This delay is necessary in order to keep the transaction from repeatedly conflicting with the same transaction.

For U-R, NU-R and U-NR, the time to flush the transaction log is also determined by the main simulation process. A FlushCheck flag system parameter determines whether the threshold criteria should be checked to determine if the flush should occur. If the FlushCheck flag is false, the flush will always occur. If the flag is true, the threshold criteria is checked against the various simulation counters. The criteria ActiveBased, CommitBased, TimeBased, and TransBased as described above are also used to describe the log flush criteria. The time to checkpoint the transaction log is also determined using these same criteria definitions.

Commit consistent checkpointing is implemented in the main simulation process, once the checkpoint threshold has been met, by suspending the release of new transactions to the active list and waiting until all of the currently processing transactions to normally terminate either though commit or rollback. Once all of the transaction activity has completed, a checkpoint record is appended to the transaction log, and the transaction suspension is removed, thus allowing the active list to fill from the transactions available for processing. Normal processing then resumes.

**Specifics of the U-R Model**: Transaction commit in the Undo/Redo model is performed once all of the data items have been manipulated. Query transactions are not committed, they are just marked as completed. The action of committing an update transaction is accomplished by writing a commit record to the transaction log, and marking the transaction as "Committing". As the recovery algorithm goes, the

transaction is technically committed at this point, but for the simulation, it is necessary to identify which transactions still have modifications contained in the cache. During the committing state, the CM is assumed to be flushing the modifications to stable storage. Since the CM is not directed to flush the cache in U-R, the transaction stays in this state until a predetermined minimal amount of simulation time elapses. After this period, it is assumed that CM has flushed the modifications to stable storage, and no undo or redo will ever be required for this transaction and transaction's effects become durable.

**Specifics of the U-NR Model**: Each data item is forced to disk, and the item is unlocked. Once all of the data items have been processed, the commit record is written to the log and the transaction state is marked committed. This model does not use the committing state because CM does flush the cache as part of the normal course of events. As with U-R, transactions that are committed, require no redo.

**Specifics of the NU-R Model:** Transaction commit with NU-R is also similar to the U-R implementation. A commit record is appended to the log and the transaction is marked as committing while the data items are processed. Each data item is unlocked, and the data item is unpinned. As with U-R, the transaction is marked as committing until enough simulation time has elapsed for the cache slots to have been flushed to stable storage. At that time the transaction state is changed to committed, which identifies the transaction as one that redo operations are not required.

**Specifics of the NU-NR Model**: The implementation of the NU-NR model differs from the other models because it does not rely on a transaction log.

## 5. 4. Common Recovery Processing Features

Database recovery is initiated after a system failure. For U-R, NU-R and U-NR, the general recovery process first simulates transferring the transaction log to memory. The log is then scanned from the last record until the first (last occurred) checkpoint record is found. Each log record is then a candidate for undo or redo based on the particular recovery algorithm. Once all of the log records are processed, the log is flushed to stable storage, and the recovery process is terminated.

In recovery, the activities of undoing and redoing individual data items are common between the three log based algorithms. In real computer systems, the underlying operating systems "know" which cache slots have been flushed to stable storage. These items do not need to be undone or redone. This knowledge of which slots have been flushed must also be simulated in the model to achieve realistic results. It is not desirable to redo all the effects of a committed transaction when in fact, all of the modifications have been flushed to stable storage. In our simulation the recovery process accomplishes this by randomly determining a number of data items for each transaction that must be undone or redone for those transactions that have recently committed or aborted.

The action of undoing a data item first determines if the action should be performed on the particular data item using the logic described above. If the undo should proceed, the BFIM, which was part of the

log record, is written to the cache, the update is forced to stable storage, and the data item is unlocked. The redo operation is performed in a similar manner. It must be first determined if the redo should take place at all. If so, the AFIM, which is part of the log record, is written to the cache, and the update is forced to stable storage.

The NU-NRCommitList is a mechanism that ensures only one transaction can commit at a time. The master bit is implemented as a global storage location that contains the identifier for one of the two primary directories. The current and shadow directories are implemented as direct access structures each constrained at DatabaseSize items. Recovery using this model is accomplished by first retrieving the master directory bit and then transferring the current and scratch directories contents to memory. A new shadow directory is allocated, and then each item is copied from the current directory to the new shadow directory. Each scratch directory is then traversed, unlocking any data items found. The new current and shadow directories are then forced to stable storage. The recovery is then completed.

## 5. 5. Simulation Models

In this section we present our simulation model for all recovery algorithms and describe the flow of transactions through it. This description is useful for simulation purpose and complements recovery algorithms given earlier. We realize some duplication of information in presenting the flow of transaction but it is necessary for a clear description of our approach. We divided the entire simulation into two parts: forward processing of transactions and database recovery from a system failure. We first explain the forward processing followed by the recovery model.

**Forward Processing in U-R:** Figure 1 presents forward processing model of U-R algorithm.

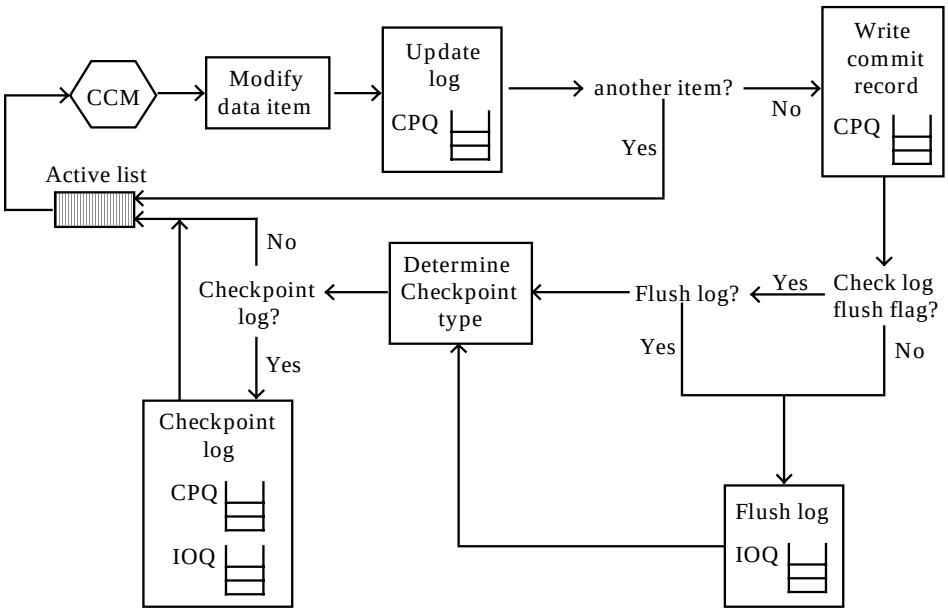


**Figure 1: U-R Model (Forward transaction processing)**

1. Select the first transaction (head of the queue) from the Active List to process.
2. For each data item in the transaction:
   a. Using the CCM, request and receive an exclusive lock on the data item.
   b. Modify the data item as described in the transaction. This is accomplished by writing the new AFIM to the cache slot of the data item.
   c. Update the transaction log with information describing the data item modification.
3. Once all data items are modified, write a Commit record to the transaction log.
4. If the log flush flag is to be checked, check it. Otherwise, flush the transaction log to stable storage.
5. If the log flush flag is checked and the conditions are met to flush the log, flush the transaction log to stable storage.
6. Determine the type of checkpoint criteria.
7. Using the checkpoint criteria, if the checkpoint threshold has been met, perform a checkpoint on the transaction log.
8. End of transaction.

**Recovery in U-R:** Figure 2 presents the recovery model of U-R algorithm.

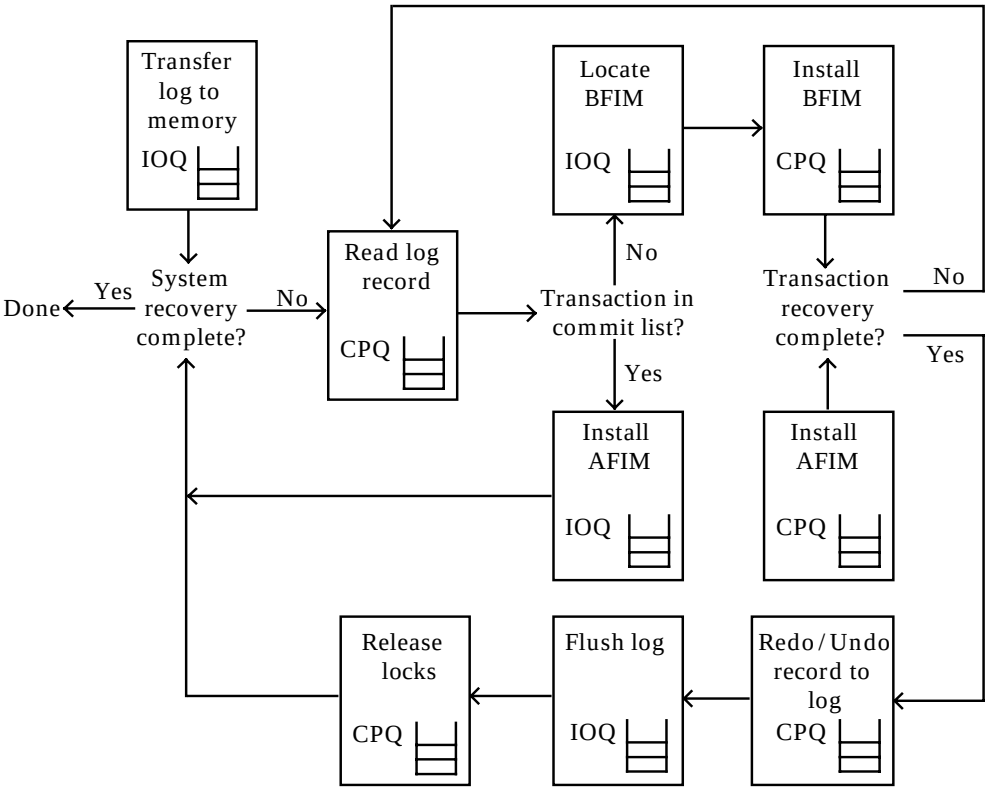


**Figure 2. Recovery model for U-R algorithm**

1. Transfer the transaction log from stable storage to memory.

2. If the system recovery is not complete, read the last (closest to the end) U-NRead log record.
3. If the transaction that wrote the log record is in the commit list:
   a. Locate the AFIM - part of the log record.
   b. Redo the modification by writing the AFIM to the cache.

Otherwise:

   c. Locate the BFIM in the log.
   d. Undo the modification by writing the BFIM to the cache.
4. If the recovery for the transaction that wrote the log record is not complete, read the next log record and repeat step 3.
5. Since the recovery for the particular transaction is complete, write an undone or redone (as applicable) record to the transaction log.
6. Flush the transaction log to stable storage.
7. Release the exclusive locks on each of the data items held by the transaction.
8. End of transaction recovery.

**Forward processing in U-NR:** Figure 3 presents forward processing in U-NR Model.

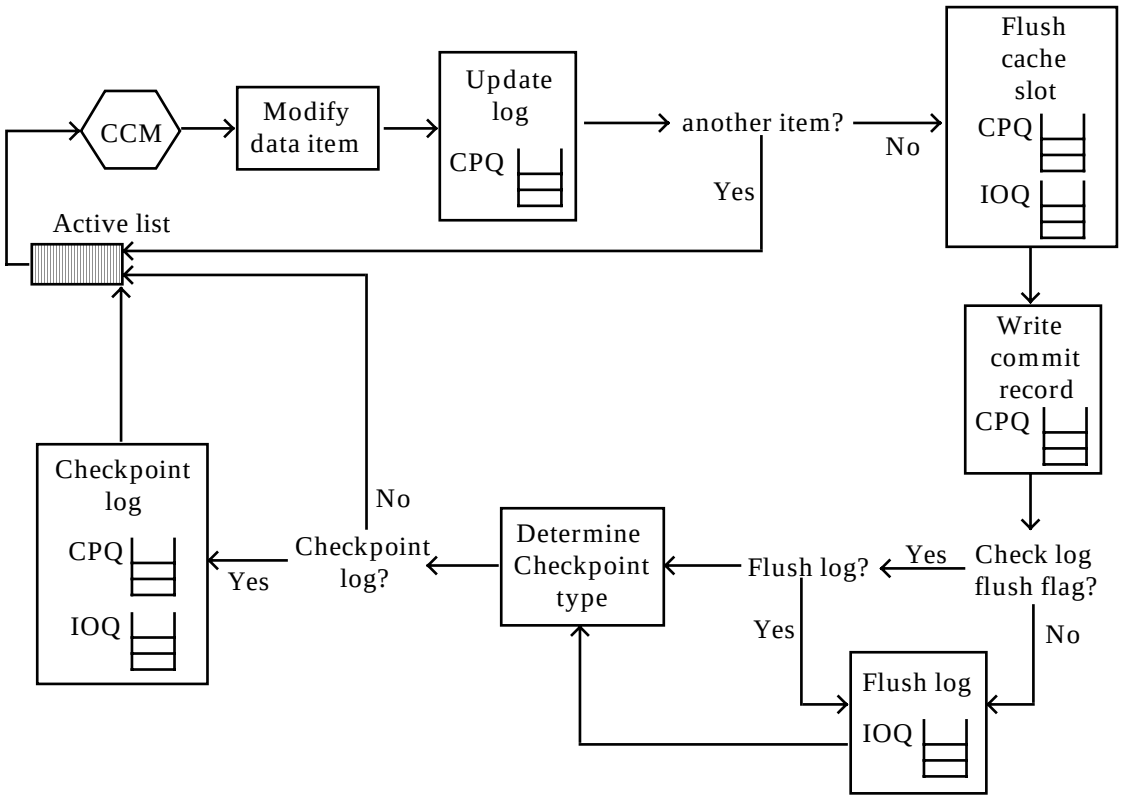


**Figure 3. Forward processing in U-NR Model**

1. Select the first transaction (head of the queue) from the Active List to process.
2. For each data item in the transaction:

a. Using the CCM, request and receive an exclusive lock on the data item.
b. Modify the data item as described in the transaction. This is accomplished by writing the new AFIM to the cache slot defined for the data item.
c. Update the transaction log with information describing the data item modification.

3. Once all data items for the transaction are modified, direct the Cache Manager to flush the cache slots containing the transaction's modifications.
4. Write a Commit record to the transaction log.
5. If the log flush flag is to be checked, check it. Otherwise, flush the transaction log to stable storage.
6. If the log flush flag is checked and the conditions are met to flush the log, flush the transaction log to stable storage.
7. Determine the type of checkpoint criteria.
8. Using the checkpoint criteria, if the checkpoint threshold has been met, perform a checkpoint on the transaction log.
9. End of transaction.

**Recovery in U-NR:** Figure 4 presents recovery in U-NR model.

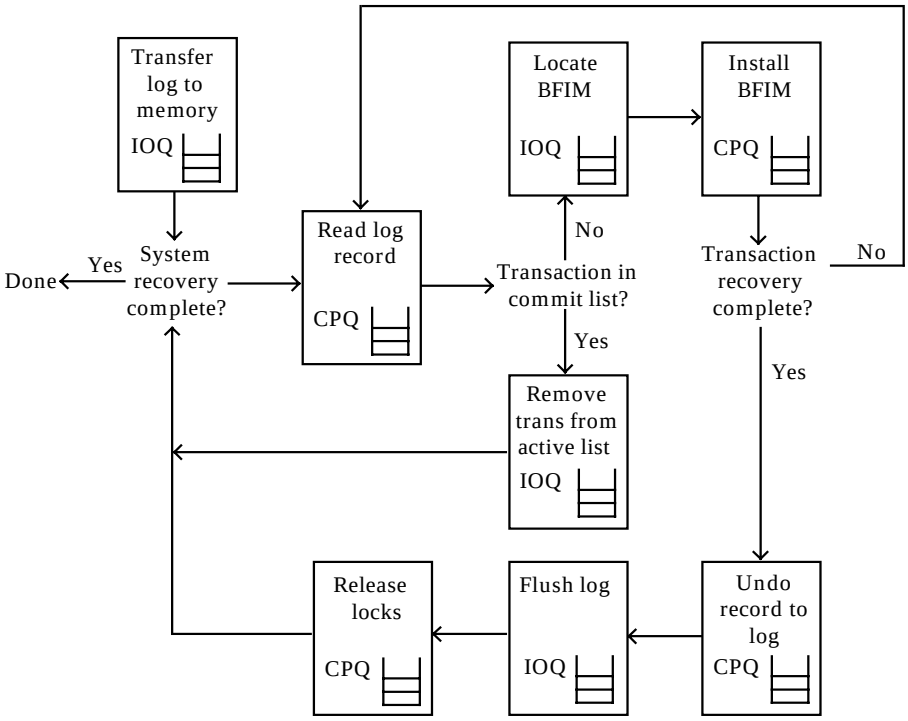


**Figure 4. Recovery in U-NR Model**

1. Transfer the transaction log from stable storage to memory.
2. If the system recovery is not complete, read the last (closest to the end) U-NRead log record.

3. If the transaction that wrote the log record is in the commit list then remove the transaction from the active list if it is there. Read the next log record and repeat step3.

Otherwise

   a. Locate the BFIM in the log.
   b. Undo the modification by writing the BFIM to the cache.

4. If the recovery for the transaction that wrote the log record is not complete, read the next log record and repeat step 3.
5. Since the recovery for the particular transaction is complete, write an undone record to the transaction log.
6. Flush the transaction log to stable storage.
7. Release the exclusive locks on each of the data items held by the transaction.
8. End of transaction recovery.

**Forward processing in NU-R:** Figure 5 presents forward processing in NU-R Model.

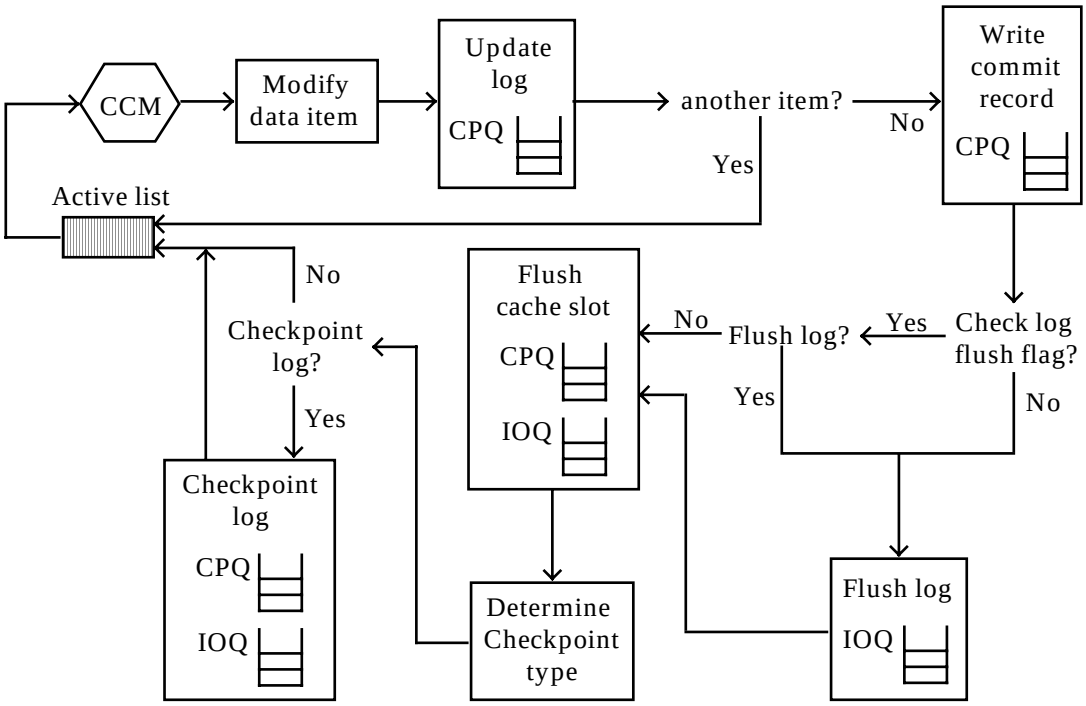


**Figure 5. Forward processing in NU-R Model**

1. Select the first transaction (head of the queue) from the Active List to process.
2. For each data item in the transaction:
   a. Using the CCM, request and receive an exclusive lock on the data item.
   b. Modify the data item as described in the transaction. This is accomplished by writing the new AFIM to the cache slot defined for the data item.

c. Pin the cache slot so that it cannot be flushed prematurely.

d. Update the transaction log with information describing the data item modification.

3. Once all data items are modified, write a Commit record to the transaction log.
4. If the log flush flag is to be checked, check it. Otherwise, flush the transaction log to stable storage.
5. If the log flush flag is checked and the conditions are met to flush the log, flush the transaction log to stable storage.
6. Unpin the cache slots containing the modifications made by the transaction, thus allowing CM to flush the slots.
7. Determine the type of checkpoint criteria.
8. Using the checkpoint criteria, if the checkpoint threshold has been met, perform a checkpoint on the transaction log.
9. End of transaction.

**Recovery in NU-R:** Figure 6 describes recovery in NU-R Model.

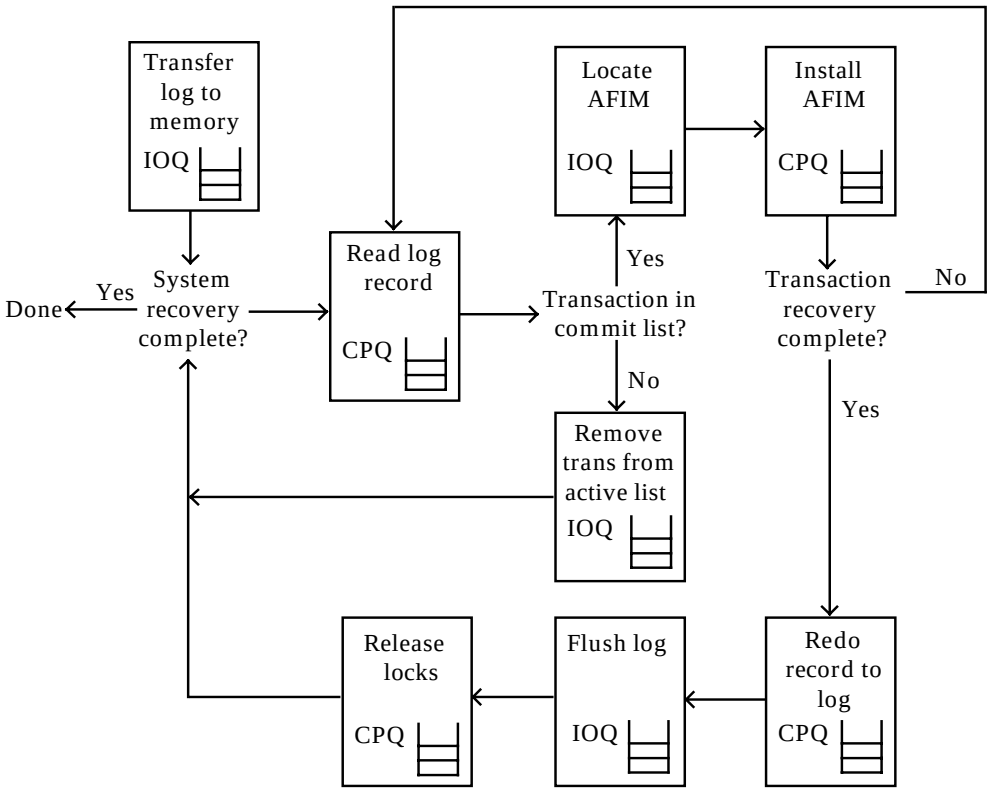


**Figure 6. Recovery in NU-R Model**

1. Transfer the transaction log from stable storage to memory.
2. If the system recovery is not complete, read the last (closest to the end) U-NRead log record.

3. If the transaction that wrote the log record is not in the commit list then remove the transaction from the active list if it is there. Read the next log record and repeat step3

Otherwise

   a. Locate the AFIM in the log.
   b. Redo the modification by writing the AFIM to the cache.

4. If the recovery for the transaction that wrote the log record is not complete, read the next log record and repeat step 3.
5. Since the recovery for the particular transaction is complete, write a redone record to the transaction log.
6. Flush the transaction log to stable storage.
7. Release the exclusive locks on each of the data items held by the transaction.
8. End of transaction recovery.

**Forward processing in NU-NR:** Figure 7 describes forward processing in NU-NR Model.

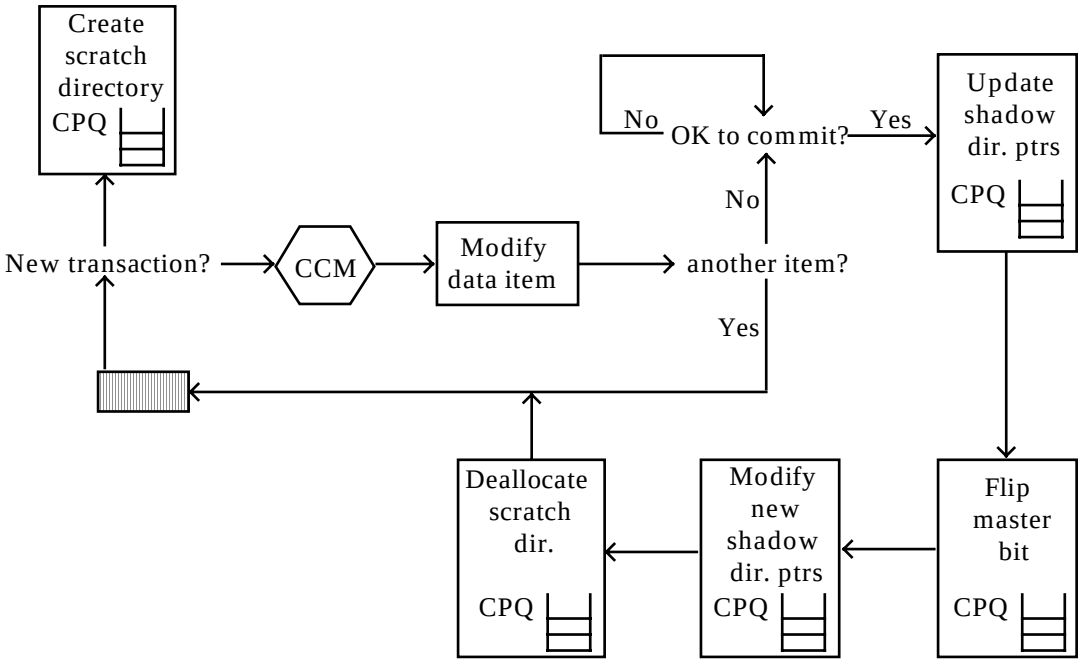


**Figure 7. Forward processing in NU-NR Model**

1. Select the first transaction (head of the queue) from the Active List to process.
2. Create a scratch directory in memory to hold values modified by the transaction.
3. For each data item in the transaction:
   a. Using the CCM, request and receive an exclusive lock on the data item.
   b. Modify the data item as described in the transaction. This is accomplished by writing the new AFIM to the scratch directory created for the transaction.

4. Once all data items are modified, determine if another transaction is in the process of committing. Only one transaction may commit at a time. Transactions desiring to commit when another is in the commit phase are queued in a first-in-first-out (FIFO) basis until the next transaction can commit.
5. Once it is OK to commit, identify the current and shadow directories. Update the shadow directory with the contents of the transaction's scratch directory.
6. Flip the master directory bit, thus committing the transaction. The master bit is stored on stable storage at a known location (fixed address). This allows the Recovery Manager to "know" where to find the information describing the master directory. The master bit information must also be forced to stable storage.
7. Modify the new shadow directory pointers to reflect the changes made in the transaction's scratch directory.
8. Deallocate the transaction's scratch directory.
9. End of transaction.

**Recovery in NU-NR:** Figure 8 illustrates recovery in NU-NR Model.

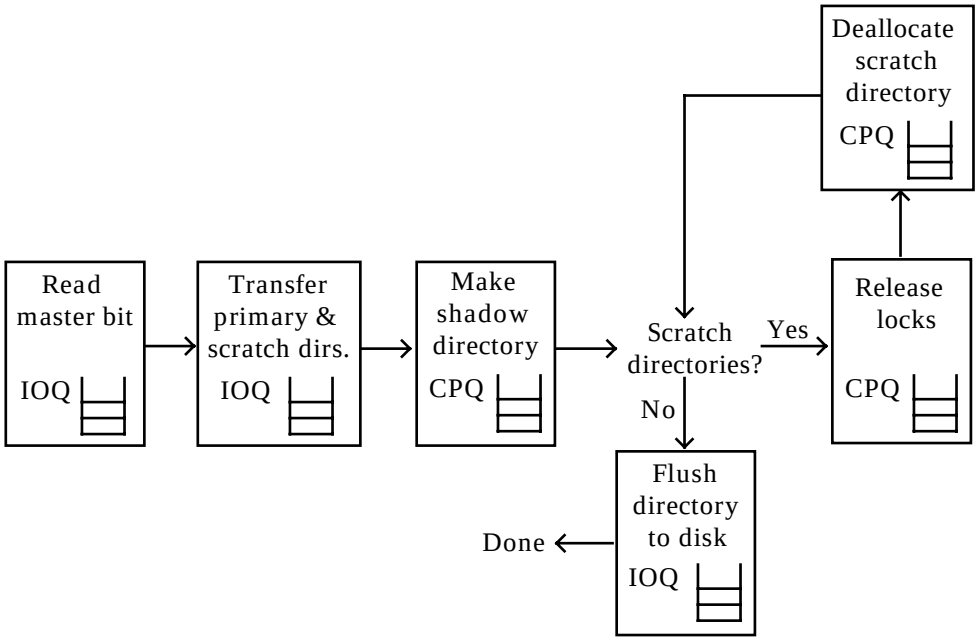


**Figure 8. Recovery in NU-NR Model**

1. Locate the Master Directory Bit at the fixed location on stable storage.
2. Transfer the directory described by the master bit and any scratch directories from stable storage to memory.
3. Create a new shadow directory by duplicating the current master directory to a new directory structure.

4. For (any) scratch directories retrieved:
    a. Release any data item locks held by transactions creating the scratch directory.
    b. Deallocate the scratch directory.
5. Flush the new master and shadow directories to stable storage.
6. Transaction recovery is complete.

## 6. Simulation Results and Discussion

The basis of the performance comparison for each of the experiments is total recovery time from a random system failure. The total recovery time is the time spent in the actual database recovery, and not the time expended in forward processing for the purposes of an eventual recovery. These values are measured in independent time units, which can represent milliseconds if desired.

With this in mind, three parameters - multiprogramming level (MPL), checkpoint threshold, and failure threshold were selected for this series of experiments. The experiments involve changing the values for these three parameters to determine the effect on overall recovery time.

The values for the parameters were determined by preliminary experimentation with the model to determine the upper and lower threshold values for each of these parameters. The constant values used for establishing the recovery time were initially set and then modified with successive runs of the model. Table 1 presents the parameter values and the timing constants used to collect recovery times.

### 6. 1. Experiment 1: The effect of the number of checkpoints on the recovery time.

The experiment is conducted by processing a set of transactions against U-R, NU-R, and U-NR models. For each model, separate simulation runs are conducted while varying the MPL values. For each model and MPL value, the transaction log is checkpointed various times before the system failure is encountered. Since MPL is defined as the number of concurrently executing transactions, it is desired to show the differences between MPL values by making the number of active transactions equal to the MPL at the time of the failure. This will ensure that the maximum number of transactions are active with respect to the MPL, and thus more transactions are susceptible to recovery processing as the MPL increases.

Number of Checkpoints ranges from 0 to 5. In order to have the desired number of checkpoints occur using CommitBased checkpoint criteria, it is necessary to determine the number of committed transactions that must occur between checkpoints. Table 2 shows the Checkpoint Thresholds for each of the MPL values that is used to ensure the desired number of checkpoints occur before the Failure Threshold is reached. For the higher MPL values, the threshold values must be reduced in order to secure the desired number of checkpoints. The failure is introduced after 1000 transactions have committed.

Table 1. Simulation parameters and their values

| Parameter | Description | Value |
|---|---|---|
| Checkpoint Criteria | Criteria under which a checkpoint is introduced | CommitBased |
| Checkpoint Threshold | Value at which a checkpoint is introduced | Dynamically determined |
| MPL | Maximum number of transactions in the system | 25, 50, 75, 100, 125, 150 |
| Failure Criteria | Criteria under which a failure is introduced | CommitBased |
| Failure Threshold | Value at which a failure is introduced | 1000 |
| Average Think Time | Average transaction delay | Mean 10 - (exp. distribution) |
| Database Size | Number of elements in the sample database | 10000 |
| No. of Transactions | Number of transactions processed in a session | 5000 |
| Transaction Size | Average number of data items in a transaction | Mean 15 - (exp. distribution) |
| W/R Percentage | Percentage of write:read | 100% (write only) |
| Check Flush Flag | Directs whether to look at Log Flush Criteria | True |
| Flush Criteria | Log flushing criteria | CommitBased |
| Flush Threshold | Value at which the log is flushed | 1 |
| Lock Time | Time required for CCM to lock a data item | 25 |
| Unlock Time | Time required for CCM to unlock a data item | 10 |
| Rollback Delay | Transaction restart delay | Mean100 - (exp. distribution) |
| Flush Delay | Cache slots to flush time estimation | Mean 1000 - (exp. distribution) |
| I/O Transfer Time | Time to transfer a unit to/from disk | 25 |
| Log Access Time | Time required to access the log | 2 |
| Log Update Time | Time required to update the log | 5 |
| Undo/Redo Time | Time required to update cache slot | 2 |

Table 2: Checkpointing thresholds.

| MPL | 1 CP | 2 CP | 3 CP | 4 CP | 5 CP |
|---|---|---|---|---|---|
| 25 | 500 | 333 | 250 | 200 | 160 |
| 50 | 500 | 333 | 250 | 200 | 150 |
| 50 | 500 | 333 | 250 | 180 | 120 |
| 100 | 500 | 333 | 250 | 160 | 120 |
| 125 | 500 | 333 | 200 | 150 | 100 |
| 150 | 500 | 333 | 200 | 120 | 80 |

Figures 9 through 14 shows the recovery times increase with MPL for different number of checkpoints. This relationship holds for all three of the algorithms at all levels of checkpointing. Since it was desired for the active count to be equal to the MPL at the time of failure, each higher MPL has more transactions to recover than the immediately preceding (smaller) MPL. A greater number of transactions to be recovered results in greater recovery time. The results also show that first recovery time reduces with number of checkpoints and then it begins to increase (Figures 15 through 20, UR-25 means UR with MPL of 25)

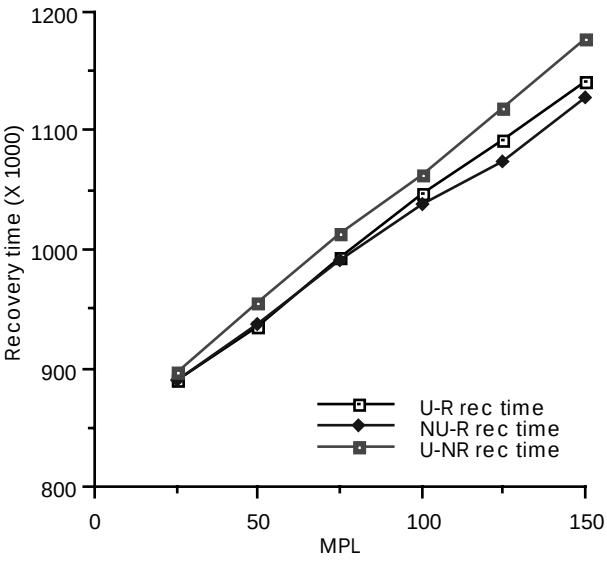


Figure 9 - Recovery Time vs. MPL

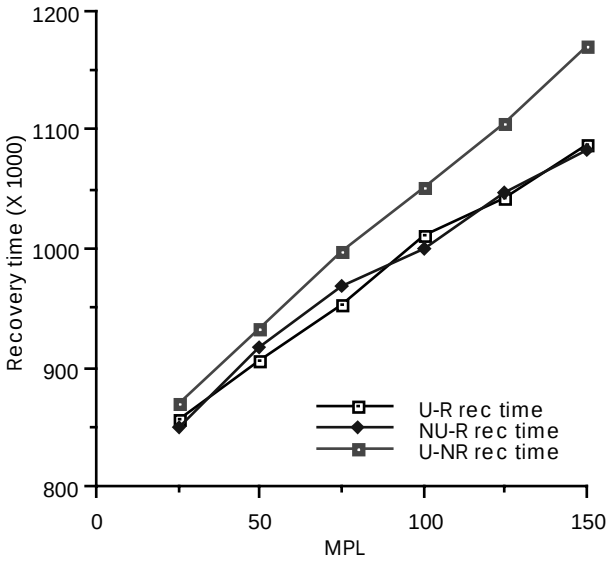


Figure 10 - Recovery Time vs. MPL

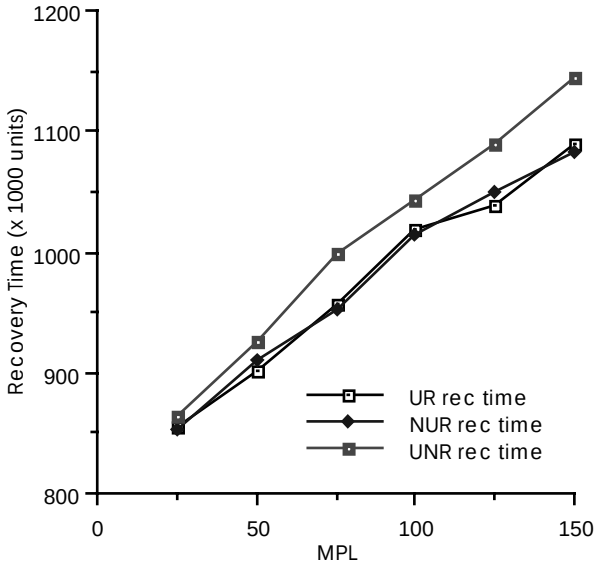


Figure 11 - Recovery Time vs. MPL

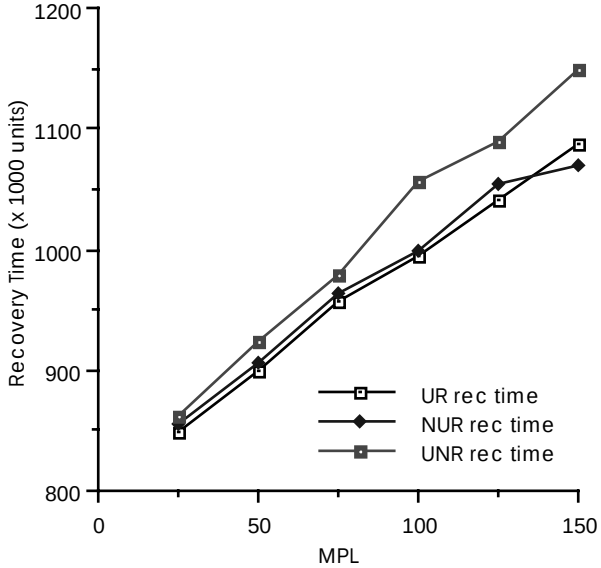


Figure 12 - Recovery Time vs. MPL

Three equally distributed checkpoints performed before the failure is introduced (at 1000 commits) delivers the best recovery times. A possible explanation is that four and five checkpoints do not continue to increase the recovery performance may lie with the transaction log. Checkpointing tends to increase the size of the transaction log. With the simulation time value for I/O transfers of 25 units, the time spent transferring the larger log from disk may outweigh the savings of processing (undoing and redoing) a smaller number of data items.

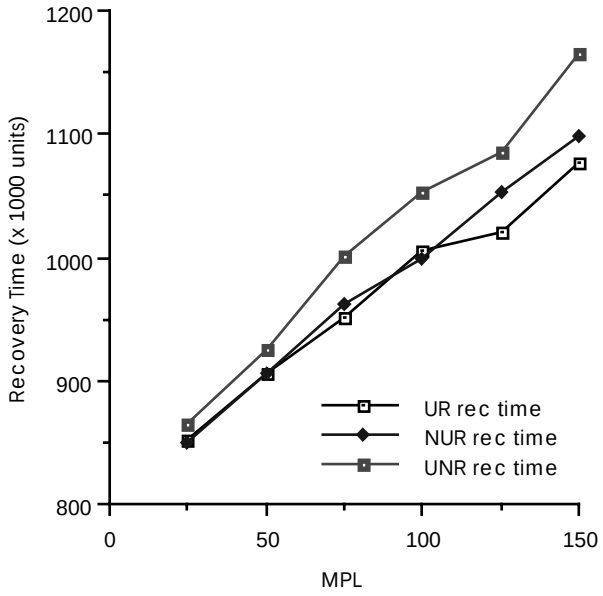


Figure 13 - Recovery Time vs. MPL

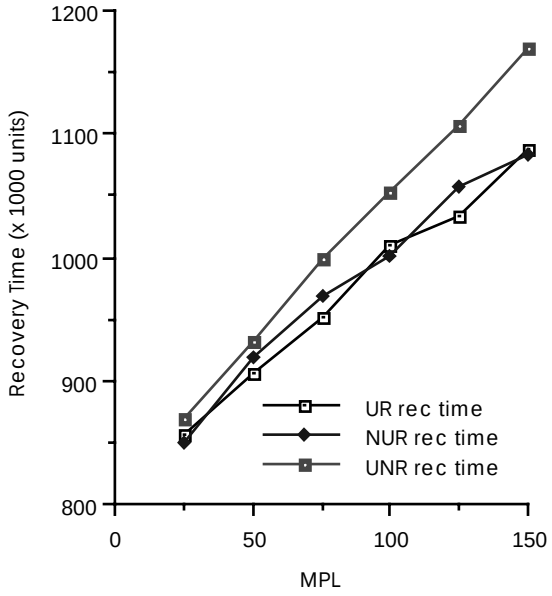


Figure 14 - Recovery Time vs. MPL

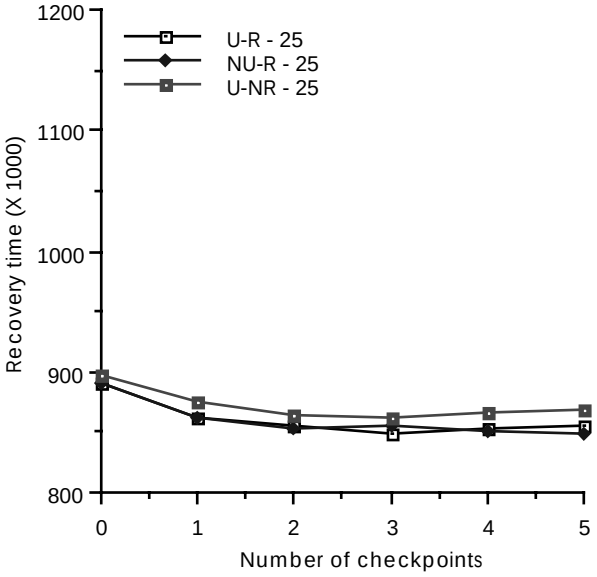


Figure 15- Recovery Time vs. Checkpoints

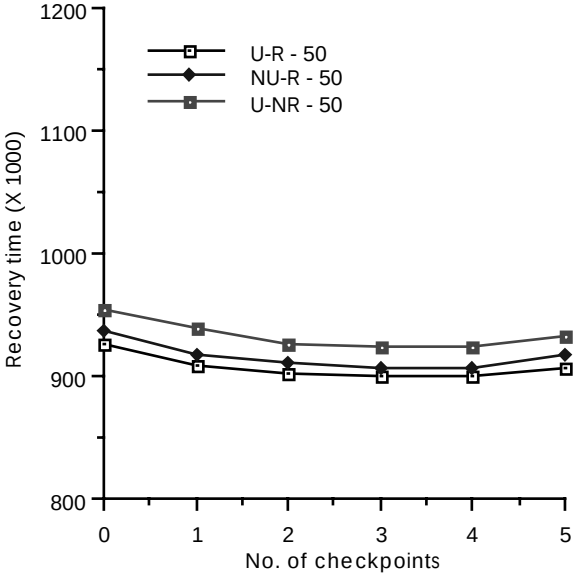


Figure 16 - Recovery Time vs. Checkpoints

This log size differential assumption can explain why the NU-R and the U-R performance is so similar when the number of transactions that must be recovered with U-R exceeds that of NU-R. In this experiment, all three of the algorithms record both BFIMs and AFIMs in the log, so this fact should have no bearing on the overall log sizes. In order to explain the results, the U-R logs must be smaller than the NU-R logs, and thus are transferred to memory faster. Once transferred however, U-R must recover more transactions, performing both undo and redo operations, than NU-R, which only performs redo operations.

The overall effect is that these two stages of the recovery operation (log transfer and transaction recovery) balance each other with respect to recovery time. Another experiment will deal with the size of the transaction log and its effect on recovery times.

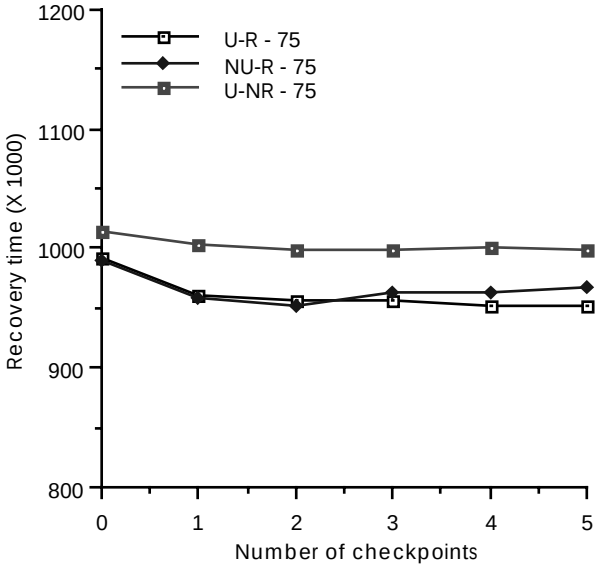


Figure 17 - Recovery Time vs. Checkpoints

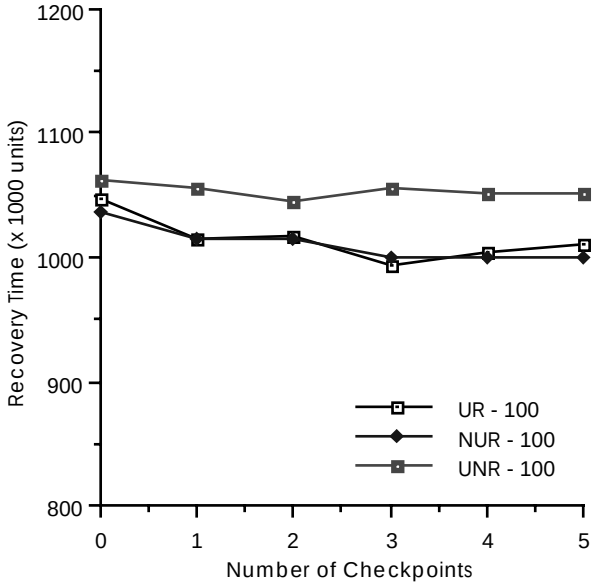


Figure 18 - Recovery Time vs. Checkpoints

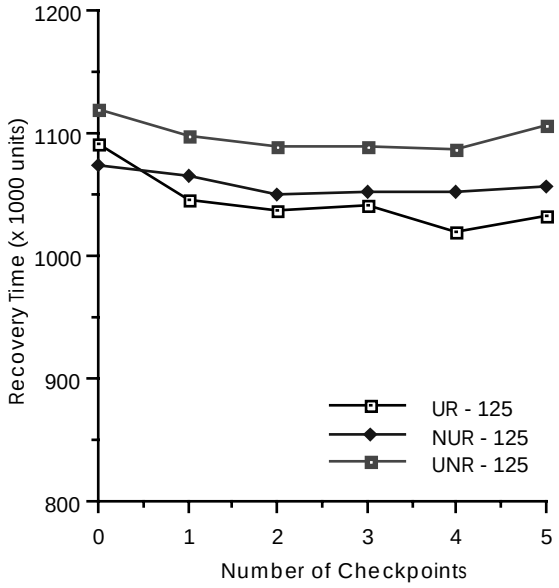


Figure 19 - Recovery Time vs. Checkpoints

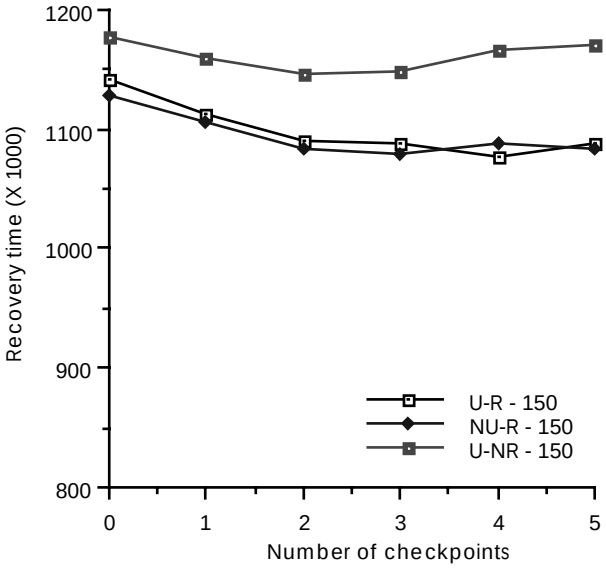


Figure 20 - Recovery Time vs. Checkpoints

The results for U-R and NU-R are very similar for all levels of checkpointing. These two algorithms consistently produce the lowest recovery times. In all situations, the U-NR recovery times are the longest. At MPL 100 (Figure 18), for example, the NU-R algorithm delivers the best performance at checkpoint

levels of 0, 1, 2, and 4 checkpoints. The U-R algorithm delivers the best recovery time at the 3 and 5 checkpoint levels. The recovery times for the U-NR algorithm range from 2.5% to 6.2% longer than the other two algorithms at MPL 100.

The gap between the U-NR recovery times and those of the other algorithms appears to widen at the highest MPL values and at the highest level of checkpointing. For example, at MPL 25 (Figure 15), U-NR ranges from 0.7% to 2.2% longer recovery times, while at MPL 150 (Figure 29), U-NR ranges from 4.3% to 8.2% longer recovery times. It can be assumed that this trend would continue as the MPL or level of checkpointing increases.

The rate of growth in the U-R recovery times appears to slow between MPL values of 100 and 125 for all levels of checkpointing. Perhaps this is the "optimum" MPL for this algorithm given the other constraints of the experiment. A possible explanation as to the reason for the comparatively poor U-NR recovery times may lie in the behavior of the algorithm, and, as described above, the length of time required to transfer the log from disk to memory. The U-NR algorithm keeps data items locked for a longer period of time than U-R or NU-R. A greater number of CCM conflicts will occur when data items are locked for a greater length of time. A larger number of rolled back transactions is the result of the CCM conflicts. Rolled back transactions must update the transaction log, thus increasing its size. With the rather high I/O transfer time in the experiment, the cost of transferring the larger logs could possibly cancel the benefits of processing a smaller number of data items as defined in the U-NR and NU-R algorithms. Another experiment will deal with the time required to transfer the log from disk and its effect on the performance of the models

## 6. 2. Experiment 2: Relationship between log size and recovery time

The intent of this experiment is to determine if there is a relationship between log size and recovery time. In the past experiment, it was speculated that the U-NR algorithm created a larger transaction log because it keeps data items locked for a longer period of time resulting in a greater number of aborted transactions. Larger logs, it was argued, in a system with a very slow I/O transfer time could have an impact on recovery time. This experiment will attempt to validate the assumption.

The experiment is conducted using the situation created for experiment 1, with the additions of measuring the size of the transaction log at the time of failure and counting the number of records that are examined upon recovery. To simplify the experiment, it is conducted keeping the MPL constant at 100. This is a median value, and from the results of experiment 1, should be able to correctly represent the model in the experiment. Table 2 describes the Checkpoint Thresholds used in the experiment. The thresholds found in the MPL=100 row are used in the experiment and the failure threshold and criteria were the same as in experiment 1.

## 6. 3. Results and Interpretation

The total log sizes are presented in Figure 21 and the number of log records processed is shown in Figure 22. Figure 21 indicates that the log sizes increase as the number of checkpoints increases. The logs produced by the U-NR algorithm range from 3.2% to 7.5% larger than those of U-R or NU-R. U-R and NU-R logs are very similar in size at all levels of checkpointing. Figure 22 shows that, as expected, the number of log records to be processed decreases as the number of checkpoints increases. Because of the scale of the graph and the fact that the U-R and NU-R results are so similar, only one set of points can be observed. U-R and NU-R consistently have a smaller number of log records to process, except in the case of two checkpoints, where all three algorithms have virtually the same number of log records to process.

Comparing the results from Figure 22 with Figure 18, there is a correlation between the number of log records processed and the total recovery time. The number of log records processed decreases as the recovery time decreases. The total log sizes for the U-R and NU-R algorithms are smaller and thus it takes less time to transfer them to memory. The recovery process requires the entire log to be transferred to memory, before it determines which records are to be processed. This means that regardless of how small the number of records to be processed, the overall recovery time must include the (rather lengthy) time required to transfer the entire log from disk.

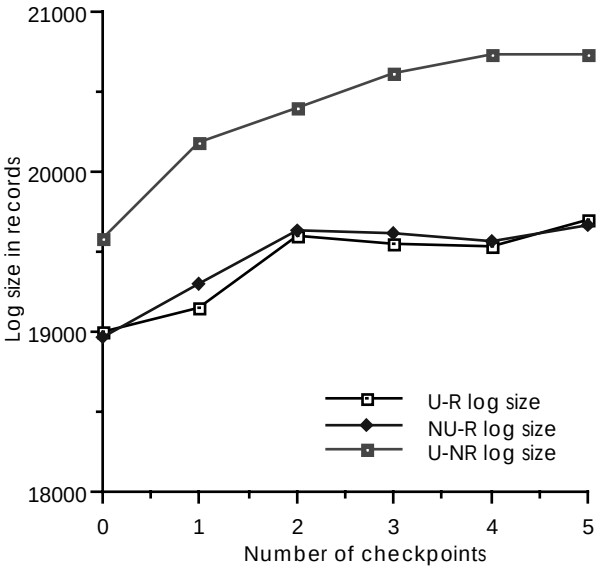


Figure 21 - Total Log Size (MPL=100)

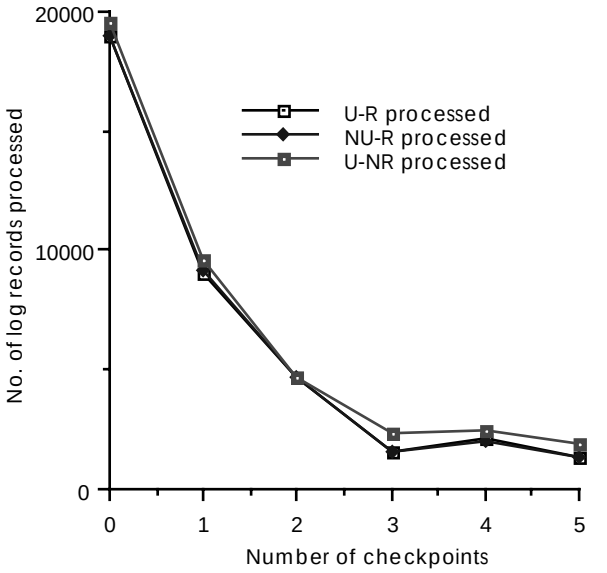


Figure 22 -Log records Processed (MPL=100)

## 6. 4. Experiment 3: Number of rolled back transactions and recovery time

The intent of this experiment is to determine if a larger number of rolled back transactions in forward processing results in a longer recovery time. In the first experiment, it was speculated that the U-NR algorithm aborted more transactions, creating a larger transaction log, which, as verified in the second experiment, should result in longer recovery times.

It was speculated that since U-NR exercises control over when a transaction is added to the commit list, the transaction's data items remain locked for a longer period of time. The extra "lock time" causes additional conflicts between transactions over the locked data items. These conflicts are resolved by the CCM by rolling back one of the offending transactions. The experiment is conducted using the situation created for experiments 1 and 2, with the addition of counting the number of transactions that must be rolled back in forward processing and MPL = 100. Table 2 describes the checkpoint thresholds used in the experiment (same as experiment 1).

## 6. 5. Results and Interpretation

The total number of aborted transactions is depicted graphically in Figure 23. U-NR shows the largest number of aborted transactions, ranging from 30% to 47% more aborts than NU-R. NU-R consistently aborts more (4.2% to 13%) transactions than U-R. Comparing Figures 23 and 18, it is possible to explain the much higher recovery times (Figure 18) for the U-NR algorithm by noting the much larger number of aborted transactions (Figure 23) for the same algorithm. The algorithm that causes the greatest number of transactions to abort in forward processing will result in the worst overall recovery performance.

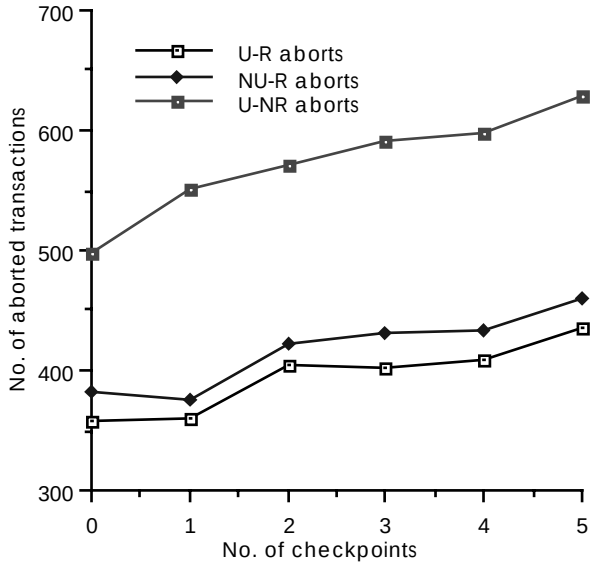


Figure 23 - Number of Aborted Transactions (MPL=100)

As expected, the NU-R algorithm aborts slightly more transactions than the U-R algorithm. The difference between these two algorithms is not great enough to cause noticeable differences in recovery time (Figure 18). In Figure 18, the NU-R algorithm delivers slightly better recovery performance at most checkpoint levels, even though it creates more aborted transactions in forward processing. This situation exists because the slightly longer I/O transfer time required to transfer the NU-R log from disk does not outweigh the greater number of data items that must be recovered with the U-R algorithm. The amount of data items that must be recovered with each algorithm will be explored in the next experiment.

## 6. 6. Experiment 4: Number of transactions in a recovery.

The intent of this experiment is to determine the number of transactions that must be recovered for each algorithm under the same circumstances. In the third experiment, it was speculated the U-R algorithm must recover a greater number of transactions than the NU-R algorithm, and thus could explain why the NU-R algorithm delivers slightly better performance than the U-R algorithm even after causing slightly more transactions to abort in forward processing (Figure 23). The experiment is conducted using the situation created for experiment 1 with the addition of counting the number of transactions that are undone and redone during recovery operations. Table 2 describes the checkpoint thresholds used in the experiment and the thresholds found in the MPL=100 row are used in the experiment.

## 6. 7. Results and Interpretation

The number of recovered transactions is depicted graphically in Figure 24 which shows the number of transactions undone and redone with the U-R algorithm, the number of redone transactions with the NU-R algorithm, and the number of undone transactions with the U-NR algorithm. It is clear from Figure 24, that the U-R algorithm processes a relatively constant 100 transactions during the recovery process. Compared to both NU-R and U-NR, the U-R must recover between 30% and 40% more transactions than the next closest algorithm.

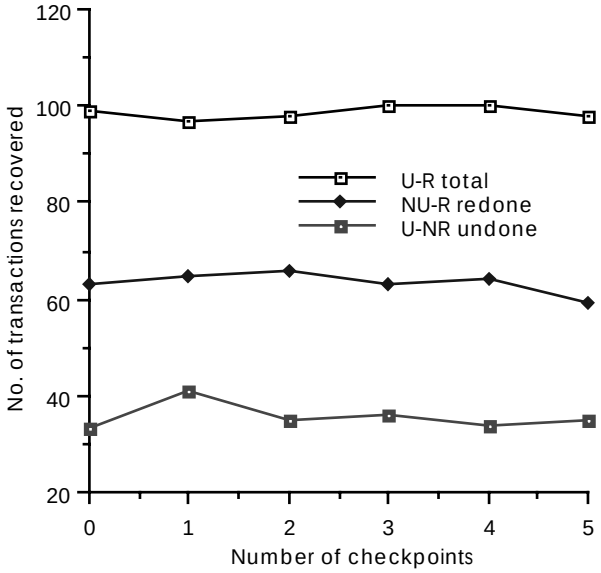


Figure 24 - Number of Recovered Transactions (MPL=100)

What is unexpected is that NU-R algorithm recovers between 59 and 66 out of the 100 transactions as compared to U-NR which only must recover between 28 and 41. The simulation constant Flush Delay (Table 2) can explain this result. Flush Delay simulates the time required for the Cache Manager to determine which cache slots to flush to stable storage during the commit process. The simulation uses a value of 1000 time units for this constant. This appears to hold the transactions a lengthy period of time

during the commit process, and thus exposing the transactions to the redo operation when the failure occurs. Since the system is fully loaded (active count = MPL), no additional transactions can become active until a transaction either aborts or is committed. Transactions appear to spend a longer time in the process of committing than they do in actual transaction processing.

The U-NR algorithm performs the least amount of recovery processing (Figure 24) and yet delivers the worst overall recovery performance (Figure 18) for MPL=100. The results obtained in Experiment 2 (Figure 21) and Experiment 3 (Figure 23) provide an explanation for this. The U-NR algorithm creates more aborted transactions, and thus a larger transaction log. The time required to transfer this larger log from disk to memory outweighs any savings gained in processing the least number of transactions during recovery.

## 6. 8. Experiment 5: I/O time and Recovery.

The intent of this experiment is to determine if decreasing the I/O transfer time, and thus increasing the I/O performance has an effect on recovery performance. It was shown in the earlier experiments that I/O performance has a significant impact on the overall recovery performance. The transaction logs maintained by all three of the algorithms must be transferred from disk to memory at the onset of the recovery process. The previous experiments showed that even though the U-NR algorithm requires less work in the form of undoing the effects of the transactions, its performance suffers because of the length of time required to transfer the log from disk. This experiment will try to determine whether a host computer system with faster I/O transfer speeds would result in different recovery performance than that presented in Experiment 1. Variable timing values were same as experiment 1, with the exception of the I/O transfer time constant. This value is reduced from 25 to 15 units. Table 2 describes the checkpoint thresholds used in the experiment and the thresholds found in the MPL=100 row are used in the experiment.

## 6. 9. Results and Interpretation

The recovery performance using the I/O transfer time value of 15 time units is shown in Figure 25. It shows that better I/O performance (lower I/O transfer times) does improve the recovery performance for all three algorithms. At three checkpoints with an I/O transfer time of 15 units, the average recovery time between the algorithms is 609. With the I/O transfer time of 25 units, the same recovery time is 1017. Reducing the I/O transfer time by 40% reduces the overall recovery time by about 40%.

The performance of the U-NR algorithm improves against the U-R and NU-R algorithms. In Figure 25, the performance of the U-NR algorithm is almost equal to that of the other algorithms, ranging from showing better performance at 2 and 4 checkpoints to 1.8% greater at three checkpoints. This performance can be compared with Figure 18, where the U-NR performance is consistently higher than the other algorithms, ranging from 1.6% to 5.7% higher than the next closest algorithm's performance.

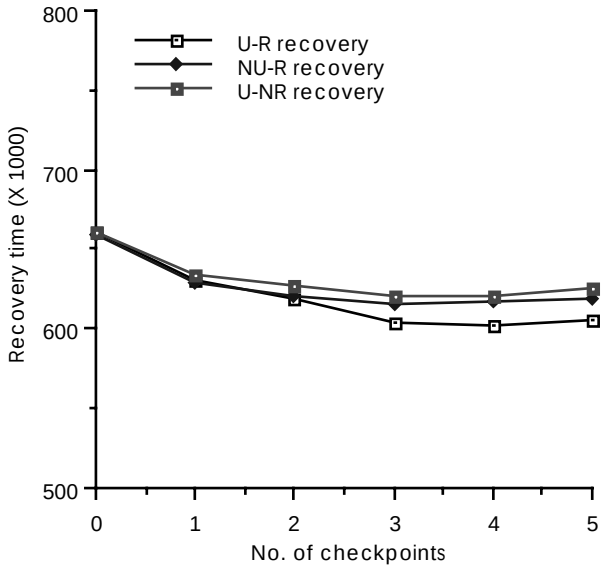


Figure 25 - Recovery Performance (I/O time 15) - MPL=100

Faster I/O performance delivers better recovery performance. In addition, after reducing the effects of transferring the log to disk, the recovery performance starts to reflect each of the algorithm's actual recovery characteristics - undoing and redoing the affected transactions. Lowering the I/O transfer rates even further should result in recovery performance reflecting the number of recovered transactions relationship presented in Figure 24.

## 6. 10. Experiment 6: Failure threshold and recovery.

The intent of this experiment is to determine if increasing the failure threshold changes the overall relationship between the recovery performance established in the earlier experiments. It is necessary to determine whether the performance described in the previous experiments reflects that of a stable database, and not fringe values captured when the database is at its limits. Increasing the failure threshold allows additional transactions to be created and flow through the system, giving it additional time to stabilize. The experiment is conducted using the situation created for experiment 1 with MPL = 100. The I/O transfer time remains at 25 time units (as opposed to 15 time units from Experiment 5). Because of the higher failure threshold, the checkpoint thresholds must be adjusted accordingly. The failure threshold is increased from 1000 committed transactions to 2500 committed transactions.

## 6. 11. Results and Interpretation

The recovery performance using the new failure threshold of 2500 committed transactions and MPL=100 transactions is depicted graphically in Figure 26. As expected, Figure 26 shows that the length of the recovery period increases when the failure threshold is increased. The relationship between each of

the algorithm's performance stays the same as with the lower failure threshold and presented in Figure 18. The U-NR consistently delivers the worst overall performance, while U-R and NU-R are very similar.

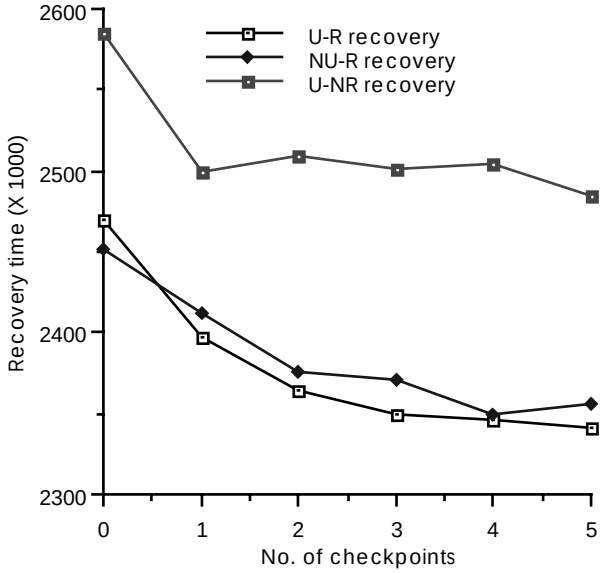


Figure 26 - Recovery Performance (MPL=100) - Failure Threshold 2500 Commits

In Figure 18, the U-NR performance ranged from 1.6% to 5.7% worse than the other algorithms. With the higher failure threshold, in Figure 26, the U-NR performance ranged from 3.7% to 6.7% worse than the other algorithms. As the failure threshold increases, U-NR will have aborted even more transactions at the time of failure, and thus created a proportionally larger log than U-R and NU-R at the time of failure. The increase in the relative recovery times for U-NR can be attributed to transferring this log from disk. It is expected that U-NR performance will get progressively worse as the failure threshold increases, given the current simulation parameter set.

## 6. 12. Experiment 7: NU-NR versus log-based algorithms.

The intent of this experiment is to provide a basis on which to compare the NU-NR algorithm with the three log-based algorithms. Checkpointing was a key component of the previous experiments, and since a transaction log is not maintained by NU-NR, it was not practical to include this algorithm in the previous experiments. This experiment is conducted in order to compare the performance of the four algorithms using the various MPL values used in the previous experiments. The experiment is conducted using the situation created for experiment 1 with the addition of the NU-NR algorithm. The log-based algorithms are configured so that there is no checkpointing of the transaction logs. For each algorithm, the MPL values are varied, and the performance for each is collected and presented for comparison. As with the first experiment, in order to fully explore the potential differences between the MPL values, it is desired to have the number of active transactions equal to the MPL at the time of failure.

## 6. 13. Results and Interpretation

The recovery performance for the MPL values 25 through 150 with no checkpoints is depicted graphically in Figure 27. It shows that the NU-NR performance is significantly worse than the other three algorithms. The NU-NR recovery times range from 41% to 46% greater than U-NR, the closet algorithm in terms of performance. The relationships between the performance of the three log-based algorithms was discussed in detail in the first experiment.

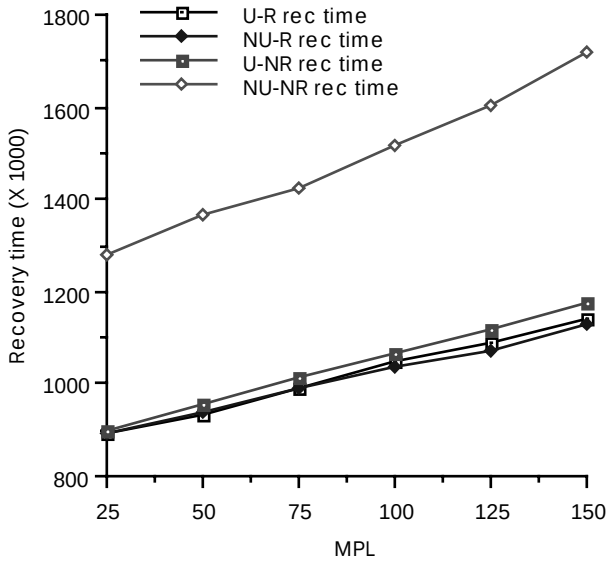


Figure 27 - Recovery Time vs. MPL (0 Checkpoints) - 4 Algorithms

The failure threshold is 1000 committed transactions, with the average number of data items per transaction set to 15 items. The logs for U-R, U-NR and NU-R should contain a minimum of 15000 records. Transferring the logs and transferring the master directory from disk to memory are similar operations, and if the log and the database sizes are equal, should take the same amount of time. NU-NR must transfer at least a 10000 item master directory from disk. Assuming that the log sizes are similar in size to the master directory, the extra work required to create the shadow directory and then reestablish the master and shadow directory structures on disk cause the NU-NR recovery performance to suffer, using the timing constants of this experiment. Once again, the I/O transfer time appears to have significant impact on overall recovery performance.

## 6. 14. Experiment 8: I/O and NU-NR .

The intent of this experiment is to determine if improving the I/O performance (decreasing the I/O transfer time) from that of experiment 7 causes the NU-NR algorithm's recovery performance to improve in comparison to the three log-based algorithms. It was assumed that the much larger I/O requirements

of the NU-NR algorithm and the slow I/O transfer time of experiment 7 explained the performance depicted in Figure 36. This experiment, as with experiment 5 was conducted between the log-based algorithms, is intended to determine if the relationships between the overall recovery performance of all four algorithms changes when I/O performance is improved. The experiment is conducted using the situation created for experiment 7, and variable timing values were same as experiment 7, with the exception of the I/O transfer time constant. This value is reduced from 25 to 15 units.

## 6. 15. Results and Interpretation

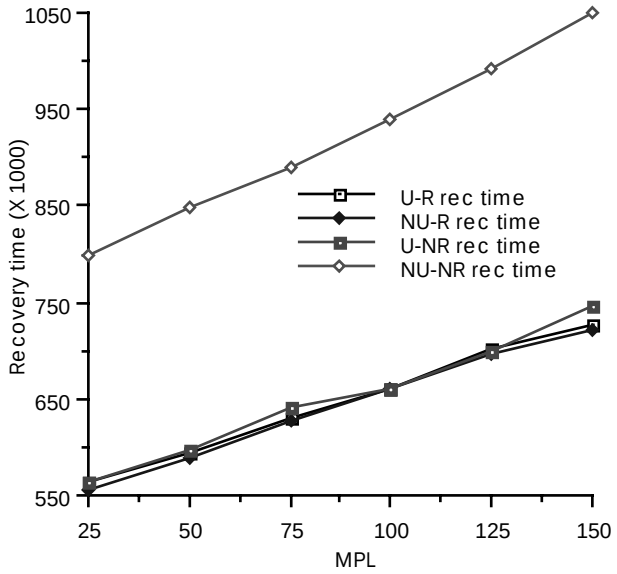


Figure 28 - Recovery Time vs. MPL (0 Checkpoints) - 4 Algorithms (I/O Transfer Time = 15)

The recovery performance for the MPL values 25 through 150 with no checkpoints and an I/O transfer time of 15 time units is presented in Figure 28. Comparing Figure 28 with Figure 27, it is clear that the recovery times decrease for all four algorithms when the I/O transfer time is lowered to 15 time units. The performance increase of the log-based algorithms was discussed fully in Experiment 5. As expected, since the NU-NR directories that must be transferred from disk are constant in size, an improvement in I/O transfer time does not improve its performance relative to the other algorithms. NU-NR recovery times exceed those of U-NR by 39% to 42%. With the longer I/O transfer time in Experiment 7, NU-NR recovery times exceeded those of U-NR by 41% to 46%. This does not appear to be a significant improvement.

In a situation where the log sizes of the log-based algorithms are significantly larger than the NU-NR directory structures, NU-NR should deliver better performance simply because of fewer I/O transfers. With no checkpoints, this situation should exist if the failure threshold is increased past the 1000 committed transactions level of this experiment.

## 7. Conclusions

The paper investigates four algorithms, Undo-Redo (U-R), Undo-No Redo (-), No Undo-Redo (NU-R), and No Undo-No Redo (NU-NR) . U-R, U-NR, and NU-R are similar algorithms in that they each maintain a stable storage log of each transaction's modifications to the database. During recovery, the logs are then examined and the transactions' effects can be undone, redone or left intact depending on the algorithm, and the state of the transaction at the time of failure. The NU-NR algorithm relies on maintaining various stable storage directory structures which describe the current database location of each data item. The recovery process using this algorithm involves restoring the last directory structure from stable storage.

The simulation results showed that, in general, the recovery times increase as the MPL increases. The results also showed that for the log-based algorithms (U-R, NU-R and U-NR), the recovery times generally decrease as the number of checkpoints increases.

The simulation results also suggest that U-R and NU-R deliver very similar performance, and in all cases, better than U-NR. We observe that U-NR delivers slower recovery times than U-R and NU-R, and that with most MPL values and across all levels of checkpointing, U-R delivers the best recovery times. It is interesting to note, that U-R must recover more transactions than either NU-R or U-NR. These results were explained by the smaller transaction log generated by the U-R algorithm and the relatively slow I/O transfer time. In the simulation, the amount of information that was required to be transferred from disk was often the key factor in the overall recovery performance.

It was shown that the U-NR algorithm causes more transactions to be rolled back in forward processing and thus creates a larger transaction log than either U-R or NU-R. As the I/O performance is improved, the performance of the U-NR algorithm improves relative to U-R and NU-R. This improved performance trend is assumed to continue as the I/O performance increases. Since U-NR is shown to require the fewest number of transactions to be recovered, once the I/O performance is improved to make it less of a factor on overall performance, the overall U-NR recovery performance could surpass that of U-R and NU-R. Increasing failure threshold does not change relationships between U-NR and other log-based algorithms.

The NU-NR algorithm delivered the worst overall recovery performance because of the large number of I/O transfers necessary to restore the current database directory in memory. Again, the speed of I/O transfers has a dramatic effect on an algorithm's recovery performance. It was speculated that this algorithm's performance could be improved by using it with a smaller database or with a system capable of much faster I/O transfer rates.

The results showed that the U-R and NU-R algorithm deliver very similar performance, and in all cases, better than U-NR or NU-NR. The slow I/O transfer rate in the simulation must account for the similarity of the U-R and NU-R recovery performance. NU-R's forward processing creates a slightly

larger transaction log than U-R. Although NU-R must process fewer transactions during recovery than UR, the savings in transaction recovery processing are offset by the time required to transfer the slightly longer log from disk. The performance of the NU-R algorithm over the U-R algorithm should be much greater if the I/O performance can be increased.

## References


[ANT92] Antani, S. "A Performance Study of Undo-Redo and Undo No-Redo Recovery Algorithms in Centralized Database SAystem", MS Thesis, University of Missouri-Kansas City, 1992.

[BER87] Bernstein, P. A., V. Hadzilacos, and N. Goodman, "Concurrency Control and Recovery in Database Systems", Addison-Wesley, 1987.

[GEL78] Gelenbe, E, and D. Derochette, "Performance of Rollback Recovery Systems under Intermittent Failure", Comm. of the ACM, Vol. 21, No. 6. 1978.

[GRI85] Griffith, N, and J. A. Miuller, "Performance Modeling of Database Recovery Protocols". IEEE Trans. on Software Eng., Vol. SE-11, No. 6. 1985.

[KEN85] Kent, J. , H. garcia-Molina, and J. Chung. "An Experimental Evaluation of Crash Recovery Mechanisms". In Proc. 4th ACM SIGACT-SIGMOD on Principles of Database Systems, Portland, March 1985.

[ORA] Oracle User Manual. Database Administrator's Guide.

[REU84] Reuter, A. "Performance Analysis of Recovery Techniques". ACM TODS, Vol. 9, No. 4, December. 1984.

[SCH89] Schwetman, H. "CSIM User's Guide". Microelectronics and Computer Tech. Co., Austin, TX.