

**A Recovery Method Supporting User-Interactive
Undo in Database Management Systems**

Won-Young Kim¹ Kyu-Young Whang²
Young-Koo Lee³ Sang-Wook Kim⁴

CS/TR-98-119

January 15, 1998

K A I S T
Department of Computer Science

This work was partially supported by the “Development of a General Purpose Multiuser Object-Storage System with Efficient Support for Variable Length Attributes(2)” project sponsored by Korea Telecom.

¹With the Computer Science Department and the Center for Artificial Intelligence Research, Korea Advanced Institute of Science and Technology (KAIST), 373-1 Koo-Sung Dong, Yoo-Sung Ku, Daejon, Korea; email: wykim@mozart.kaist.ac.kr; fax: +82-42-869-3510.

²With the Computer Science Department and the Center for Artificial Intelligence Research (Professor/Director, Database and Multimedia Lab.), Korea Advanced Institute of Science and Technology (KAIST), 373-1 Koo-Sung Dong, Yoo-Sung Ku, Daejon, Korea; email: kywhang@mozart.kaist.ac.kr; fax: +82-42-869-3510.

³With the Computer Science Department and the Center for Artificial Intelligence Research, Korea Advanced Institute of Science and Technology (KAIST), 373-1 Koo-Sung Dong, Yoo-Sung Ku, Daejon, Korea; email: yklee@mozart.kaist.ac.kr; fax: +82-42-869-3510.

⁴With the Information and Telecommunication Engineering Department, Kangwon National University, Chuncheon, Kangwon-Do, Korea; email: wook@cc.kangwon.ac.kr; fax: +82-361-250-6390.

A Recovery Method Supporting User-Interactive Undo in Database Management Systems

Won-Young Kim* Kyu-Young Whang† Young-Koo Lee‡
Sang-Wook Kim§

Abstract

User-interactive undo is a kind of recovery facility that allows users to correct mistakes easily by canceling and reexecuting operations that have already been executed. Supporting user-interactive undo is essential for authoring processes in new database applications such as software engineering, hypermedia, and computer-aided design. A partial rollback using savepoints supported by commercial database management systems (DBMSs), which allows only cancellation of executed operations, is a restricted form of user-interactive undo. Although many applications use DBMSs, they have to provide user-interactive undo by themselves due to lack of support from the DBMSs. Since implementation of user-interactive undo is quite complex, it poses significant burden to application programmers.

This paper proposes a new recovery method facilitating user-interactive undo in DBMSs. Such a facility relieves the programmers of implementing user-interactive undo themselves in developing DBMS applications. The method guarantees fast rollback of transactions that contain user-interactive undos. It also provides users with the bulk undo operation that restores the database to a predetermined point in the past. The bulk undo operation resembles partial rollback, but differs in that it allows a redo that cancels the bulk undo. Moreover, the performance of the method is comparable to that of the traditional recovery method in spite of added functionalities.

This work was partially supported by the “Development of a General Purpose Multiuser Object-Storage System with Efficient Support for Variable Length Attributes(2)” project sponsored by Korea Telecom.

*With the Computer Science Department and the Center for Artificial Intelligence Research, Korea Advanced Institute of Science and Technology (KAIST), 373-1 Koo-Sung Dong, Yoo-Sung Ku, Daejeon, Korea; email: wykim@mozart.kaist.ac.kr; fax: +82-42-869-3510.

†With the Computer Science Department and the Center for Artificial Intelligence Research (Professor/Director, Database and Multimedia Lab.), Korea Advanced Institute of Science and Technology (KAIST), 373-1 Koo-Sung Dong, Yoo-Sung Ku, Daejeon, Korea; email: kywhang@mozart.kaist.ac.kr; fax: +82-42-869-3510.

‡With the Computer Science Department and the Center for Artificial Intelligence Research, Korea Advanced Institute of Science and Technology (KAIST), 373-1 Koo-Sung Dong, Yoo-Sung Ku, Daejeon, Korea; email: yklee@mozart.kaist.ac.kr; fax: +82-42-869-3510.

§With the Information and Telecommunication Engineering Department, Kangwon National University, Chuncheon, Kangwon-Do, Korea; email: wook@cc.kangwon.ac.kr; fax: +82-361-250-6390.

1 Introduction

Software development[9], hypermedia authoring[4][16], and CAD[10] have recently become new database applications as they need to handle increasingly large volumes of data. Trial-and-errors frequently occur in the authoring process of these applications due to users' mistakes or unexpected results of operations. Thus, these applications need a recovery facility for canceling the operations already executed and even for reexecuting them.

In this paper we define *user-interactive undo* as a recovery facility[14][2][15][1] that enables users to cancel or reexecute under users' control the operations that have already been executed. User-interactive undo consists of two operations: undo and redo. The *undo operation* restores user's data to the previous state by canceling an executed operation. We call the canceled operations *undone operations*. While in a broad sense the *redo operation* is an operation that reexecutes the previously executed operation, in this paper, we restrict the redo operation to the one that reexecutes an undone operation.

Although many database management systems(DBMSs) already have a recovery facility that can recover databases from various failures by treating the entire authoring process as a transaction, they do not support user-interactive undo. A total transaction rollback is inappropriate for handling users' trial-and-errors since it forces other operations, which bear no relation to the errors and may have executed for a long time, to roll back as well. This necessitates a mechanism that cancels parts of the executed operations and reexecutes the undone operations upon the user's requests without a total transaction rollback. The recovery methods[11][6][5] used in commercial DBMSs provide such partial rollback using savepoints[11][5]. A savepoint can be established at any transaction execution point. After executing for a while, users can request canceling all the updates performed after the savepoint when they meet errors or unexpected results. After such a partial rollback, the transaction can resume normal execution. However, this partial rollback facility has a critical limitation that it cannot reexecute the undone operations, and thus, it is not usable in user-interactive undo.

Currently, since DBMSs do not support user-interactive undo, the applications do it themselves. It is complex and difficult to implement user-interactive undo since it should handle a variety of operations that may be encountered in an application. Moreover, if an application uses a DBMS, much more overhead is incurred for handling the data the DBMS itself updates. Therefore, providing user-interactive undo in the application that uses a conventional DBMS imposes severe overhead to the programmer. Since applications requiring user-interactive undo have recently been increasing, a mechanism supporting user-interactive undo directly in the DBMS would be essential.

Until now, the research on user-interactive undo has mainly focused on its models. The models define the order and the execution mechanism of their undo and redo. Typical examples of user-interactive undo models are the linear undo model[14][2], history undo model[14], and script model[15][1]. However, there has never been an approach supporting user-interactive undo directly in a transaction processing system such as a DBMS.

In this paper, we propose a new recovery method with which a DBMS can directly support user-interactive undo. In particular, we provide this additional facility without seriously altering the existing one such as crash recovery used in commercial DBMSs. We choose the history undo model[14][2] as our undo model since it has a nice property that

it can rollback to any previous state.

The proposed method has the following characteristics: 1) it provides both (repeated) undo and redo for executed operations in a DBMS while the partial rollback facility provides only undo operations; 2) it requires little overhead, which enables the performance of our method to be comparable to that of the traditional recovery method that only supports partial rollback; 3) it fully utilizes the base structure for supporting crash recovery, which makes it easy to implement the user-interactive undo in the DBMS; 4) it can reduce the implementation overhead of application programmers because user-interactive undo could be built easily using this facility provided in the DBMS.

The paper is organized as follows. In Section 2 we survey three basic user-interactive undo models. In Section 3 we discuss issues in supporting user-interactive undo in a DBMS. In Section 4 we present a new recovery method for supporting user-interactive undo and analyze its performance overhead. In Section 5 we summarize and conclude the paper.

2 User-Interactive Undo Models

User-interactive undo models define the order and the execution mechanism of their undo and redo operations, which are tightly related to their data structures. This section briefly reviews three typical user-interface undo models: the linear undo model[14][2], history undo model[14], and script model[15][1]. We present the data structures, mechanisms, benefits, and shortcomings of each model.

The linear undo model maintains two lists: a history list and a redo list. The history list keeps a sequence of the executed operations that remain in effect, i.e., that either have been executed and not yet undone or have been redone. The redo list keeps a sequence of operations that have been undone. The undo operation starts with the last operation in the history list. When an operation is undone, it is removed from the history list and put into the redo list as the first item. The last operation put into the redo list can be redone and again be appended to the history list. The undo and redo are meta operations in the linear undo model and, unlike normally executed operations, do not appear in the history list.

Figure 1 shows an example of a history list and a redo list maintained by the linear undo model. In Figure 1, C_i represents one of executed operations and q_i one of database states. When q_4 is the current state, its left is the history list and its right the redo list. At q_4 , the first operation that can be undone is C_4 , the last operation of the history list, and the first operation that can be redone is C_5 , the first operation of the redo list.

A shortcoming of the linear undo model is that users cannot recover a certain previous state in some cases. If an operation O_1 is undone and some other operations are newly executed, it is impossible to recover the previous state in which O_1 was done using only undo and redo operations in this model. For example, in Figure 1, one cannot recover state q_5 if new operation C_8 were executed at q_4 . Because undoing C_8 makes C_8 to be the first operation of the redo list, it is impossible to redo C_5 without redoing C_8 first.

The history undo model keeps a time-ordered sequence of all the operations including undo and redo on its history list. This model treats undo and redo as any other normally executed operations. Undoing an operation appends its inverse operation to the history list. There is an undo pointer that indicates an operation to be undone. While undo

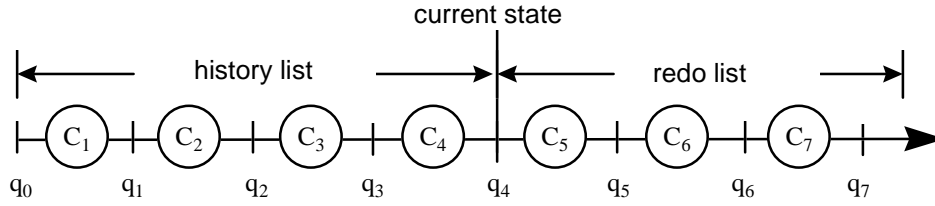


Figure 1: A history list and a redo list in the linear undo model.

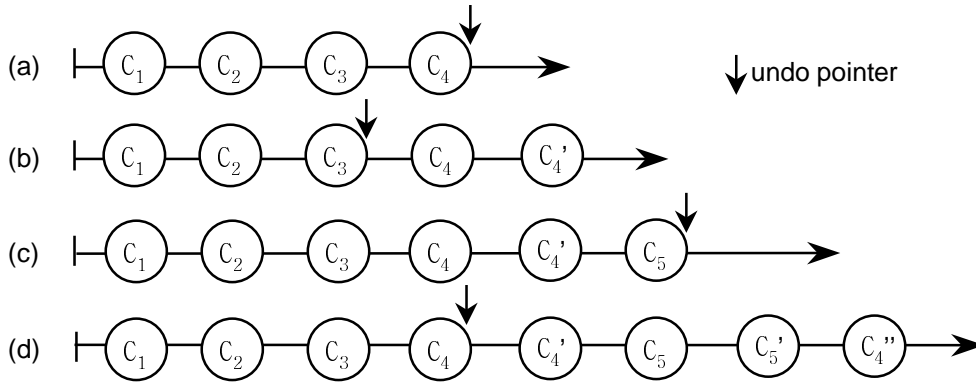


Figure 2: Undo steps of the history undo model.

operations are in progress, the undo pointer indicates the previous operation of the most recently undone one in the history list; otherwise, it indicates the last operation of the history list. Redoing an operation is undoing its inverse operation.

Figure 2 shows undo steps in the history undo model. C'_i represents the inverse of C_i , and C''_i the inverse of C'_i , i.e., the redo operation of C_i . Given the history list in Figure 2(a), suppose that C_4 is undone. Then, the resulting history list will be as in Figure 2(b), where C'_4 is the inverse operation that cancels the effect of C_4 . The undo pointer indicates C_3 , the previous operation of the most recently undone one, C_4 . If one breaks out of the undo mode by doing a new normal operation, say C_5 , the history list will become the one in Figure 2(c). At this point, doing two more undo operations will result in Figure 2(d), where the first one is an undo operation of C_5 and the second one a redo operation of C_4 .

The history undo model has the nice property that it is possible to go back to any previous state. Because its history list keeps all the ever executed operations, users can recover any previous state by undoing all the operations executed since that state. However, the history undo suffers from a quite long history list resulting from the repetitive undo and redo operations. When an operation has been undone and then redone, one has to undo the redone operation and redo the undone operation to restore any previous state of



these. This repetition makes users confused and the system spend more time to recover the previous state. We call this problem *the repetition problem* of the history undo model.

In the script model, user operations are maintained in a script file. The system executes the script file and then shows its result to users. This model supports the undo operation by allowing users to edit the script file. To undo the executed operations, one deletes them from the script file and reruns it from the initial state.

The script model is simple and powerful. However, it is not appropriate for a user-interactive interface for the following two reasons. First, its usage is inconvenient and complex because the user has to manage the process of editing and running the script file. Second, it spends considerable execution time rerunning all the operations in the script file for every interaction without a partial execution mechanism[15].

3 User-Interactive Undo in the DBMS

Research on user-interactive undo so far has focused only on the models[14][2][15]; there has been no research on integrating the interactive undo with a transaction processing system such as a DBMS. In this section, we present issues in supporting user-interactive undo in a DBMS.

First, we should select the user-interactive undo model for the DBMS operations. Here, we do not design a new one, since there are many basic studies on the user-interactive undo models as described in Section 2. Instead, we choose the history undo model for the following reasons: first, it has an advantage that a user can go back to any previous state; second, its history list has a structure very similar to the log used in a recovery method. We can utilize the log as the history list without creating a new data structure, and this makes it easy to integrate the user-interactive undo with a DBMS.

Second, we should solve the repetition problem of the history undo model. As described in Section 2, repetition of undo and redo operations makes the history list quite long. Whenever a transaction containing the undo and redo rolls back or a user returns to a desired previous state, the longer history list makes the execution take much more time. The transaction rollback time affects not only total throughput of the system but also users' waiting time. In this paper, we present two solutions to the repetition problem in the history undo model: the first is an algorithm guaranteeing fast transaction rollback in spite of repeated undos and redos, and the second a bulk undo operation by which a user can restore the database to a predetermined state with just one interaction. We describe these features in detail in Section 4.2.2.

Third, we should consider how to implement the history list. As mentioned before, we utilize the log used in the traditional database recovery method as a base structure for the history list without creating a new one. Since the history list keeps user operations (or user commands) while the log keeps internal DBMS operations, there is a one-to-many mapping between the two types of operations. User operations are DBMS calls by which users access the DBMS, and the internal DBMS operations are low-level ones performed in a DBMS and invoked by each DBMS call.

In order to specify the mapping between these two kinds of operations, we employ *boundary log records*. We can easily identify all the log records written for one user operation by enclosing them using two boundary log records. Therefore, the undo of a user operation

makes the log records between the boundary log records rolled back atomically. To avoid confusion and for easy presentation, we assume that one user operation corresponds to one log record.

Fourth, we should consider whether the undo of an operation has to release the locks acquired during its execution. Let S_b be a transaction state before the operation O is executed, and S_u be another transaction state after the operation O has been executed and subsequently undone. Assume that during the execution of the operation O , a lock L was acquired to update an object o . To make S_u be the same state as S_b , L should be released after the undo operation. However, if L is released, redoing O becomes impossible if another transaction has acquired L and updated o . Therefore, the acquired lock should be kept after the undo operation. Thus, our undo operation restores only the data values without releasing the acquired lock, and this also concurs with the traditional two-phase locking(2-PL) protocol[3].

Finally, we should provide user-interactive undo without serious changes to the traditional recovery methods, which require severe overhead in their designs and implementations. Since the proposed method utilizes the recovery data used in the traditional recovery facilities, its implementation does not require severe changes in the traditional ones. In this paper, we describe the method as an extension of ARIES[11], a well-known recovery method.

4 A Recovery Method Supporting User-Interactive Undo in a DBMS

In this section, we present a new recovery method supporting user-interactive undo in a DBMS. The proposed method extends ARIES[11], which is widely known to be a correct and reliable recovery method. In Section 4.1, we present an overview of ARIES and describe the partial rollback facility that supports only undo operations. In Section 4.2, we describe the data structures and algorithms of our method in detail. In Section 4.3, we analyze the performance of the proposed method.

4.1 ARIES Recovery Method

Overview of ARIES To achieve consistency of a transaction, ARIES records the progress of a transaction and its update actions in a log. The log consists of log records. Each log record is assigned a unique log sequence number(LSN) — the address of the corresponding log record. Every data page has an LSN field, called PageLSN, for keeping the LSN of the last update operation on itself.

ARIES records not only the updates performed during forward processing of transactions using normal log records but also the updates performed during partial or total rollback of transactions using compensation log records(CLRs)[11]. The normal log records contain both undo and redo data. The undo data provide information on how to undo the changes made by the transaction, and the redo data on how to redo them. Therefore, the operation logged using the normal log record can be either undone or redone. To make transaction rollback efficient, all the log records written by the transaction are linked via the PreviousLSN field of the log records in reverse chronological order. The PreviousLSN

Type
PreviousLSN
UndoNextLSN
RedoData

Figure 3: The data structure of a CLR.

field contains the LSN of the preceding log record written by the same transaction.

In ARIES, the update written by a CLR is never undone, and hence, a CLR contains only redo data. Figure 3 shows the data structure of a CLR. The UndoNextLSN field contains the LSN of the next log record to be undone during rollback of a transaction; this is the value of the PreviousLSN field of the log record that has just been undone. The RedoData field contains redo information for the CLR, i.e., undo information, for the corresponding update operation. The Type field indicates whether this log record is a CLR or not. When a CLR appears during transaction rollback or restart recovery, its UndoNextLSN field is used to determine the next log record to be undone.

In ARIES, restart recovery consists of the analysis pass, redo pass (or more specifically, repeating history), and undo pass. The analysis pass determines the starting point of the redo pass and finds loser transactions by scanning log records from the last available checkpoint log record[7] up to the end of the log. The loser transactions are those that were in progress when the crash occurred and will be rolled back in the undo pass. During the redo pass, ARIES repeats the operations in its log whose effects were not reflected on the database disk before failure of the system. If the LSN of a log record is greater than the PageLSN of the corresponding data page, i.e., if the effect of the log record is not reflected on that page, the log record's update should be redone. Otherwise, i.e., if the log record's update is already reflected in that page, the log record is skipped. The undo pass rolls back all the loser transactions. During this pass, undo actions for the log records belonging to the loser transactions are done in the reverse chronological order, and CLR's for the undo actions recorded in the log.

ARIES supports both page-oriented undo and logical undo for transaction rollback [11][12]. Page-oriented undo occurs when a page containing data updated during forward processing still contains the data that is about to be undone. Logical undo occurs when the page containing the data to be undone is different from the one originally modified during forward processing. This situation can happen because, in a multi-user environment, uncommitted updates of one transaction can be moved to a different page by another

transaction. For example, updated index entries are moved upon index node split. If the former transaction rolls back, logical undo occurs since the original page does not contain the data anymore. Logical undo has an overhead of accessing meta data such as indexes or system catalogs to search the page that contains the moved data. If the system were restricted to do only page-oriented undo, the latter transaction would have to wait for the former to commit. This waiting of the transactions that update the same page degrades concurrency levels of the system. Therefore, ARIES supports logical undo for high concurrency.

Unlike undo, ARIES supports only page-oriented redo because it can handle all the cases of the redo pass[11][12]. Page-oriented redo occurs when a page containing data, updated during forward processing, still contains the data which is to be redone during the redo pass. In contrast, logical redo occurs when the page containing the data to be redone is different from the one originally modified during forward processing. Performing only page-oriented redo makes the redo pass efficient since it accesses only the pages originally updated during the forward processing without accessing any meta data.

Partial rollback The partial rollback[6][11][5] is a facility that can rollback part of a transaction. Unless a DBMS provides this facility, a transaction must be rolled back in its entirety even when errors occur in its small portion. These errors are caused by deadlocks, requests for nonexistent data, or unauthorized data accesses, etc. Also, sometimes, users may want a transaction to roll back when they are not satisfied with the result of operations.

Savepoints[5] must be established before a partial rollback; these are landmarks indicating the points to which a transaction can rollback. When a savepoint is established, the LSN of the latest log record written by the transaction, called SaveLSN, and the current state of the transaction are stored in virtual storage; then, the identifier of the savepoint is returned to the user. The current state of the transaction in progress includes locks, cursors, and accessing information for volumes and files[11].

During partial rollback to a savepoint, log records are undone in the reverse chronological order and the state saved for that savepoint is restored. For each undo action, a CLR is recorded. As described in the previous section, ARIES never undoes CLR during a rollback using the chain of the CLR via the UndoNextLSN field. When a transaction is partially rolled back to a savepoint, the locks and other data structures obtained after that savepoint are released. Thus, the aborted portion of the transaction is invisible to users after the partial rollback.

A particular savepoint is no longer outstanding if another partial rollback has been performed to a preceding one. Figure 4 shows a number of savepoints established for a transaction. When C_5 was executed, the outstanding savepoints are S_0 , S_1 , S_2 , and S_3 . After a partial rollback to S_2 , S_1 is outstanding but S_3 is not, and the aborted part of the transaction containing C_3 , C_4 , and C_5 is regarded as never have been executed.

Directly applying the partial rollback scheme to user-interactive undo has the following problems. First, although the partial rollback prevents the total rollback, it does not support redo operation for the aborted part of the transaction. That is, user interaction is very restrictive since the partial rollback provides only undo operations, but not redo operations. Second, users must always be aware of the savepoints for undo. They have to establish savepoints before a partial rollback or identify the savepoints established by the

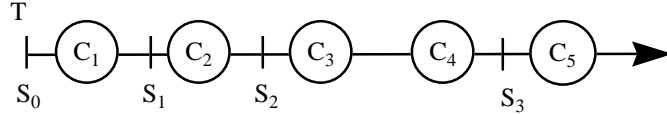


Figure 4: Savepoints in a transaction.

system, and be aware of the outstanding savepoints when they need a partial rollback.

4.2 A new recovery method supporting user-interactive undo

In this section we describe a new recovery method supporting user-interactive undo. In Section 4.2.1, we identify the characteristics of log records for undo and redo operations and propose new log record types that satisfy these characteristics. In Section 4.2.2, we describe recovery algorithms that support undo and redo operations. We describe our method as an extension of ARIES described in Section 4.1.

4.2.1 New log record types for undo/redo

To implement the history list we should devise a new type of log record for logging undo and redo operations in the history undo model. Such log records must have the following characteristics: first, although they have to contain the undo and redo information for a transaction rollback and crash recovery, they should avoid duplicates of the same information. If a system supports only page-oriented undo, the undo information of undo operations is the same as the redo information of redo operations, and vice versa. Therefore, in this case, keeping both undo and redo information is duplicating the same information, which causes severe space overhead considering that undo and redo operations are frequently executed in authoring processes. Second, they have to contain status information to identify whether the logged operations remain in effect in the database. By skipping the operations that are already undone, we can minimize the processing overhead of rollback caused by the problem in the history undo model described in Section 3.

The existing log record types used in ARIES are not applicable to user-interactive undo since they do not have the above characteristics. The CLR of ARIES is not appropriate for logging undo and redo operations because of the absence of undo information. Since normal log records contain both undo and redo information for logged operations, these can be used for logging undo and redo operations. However, these require large log space overhead due to the replicated information and also suffer from the repetition problem of the history undo model. Figure 5 shows the processing overhead at the time of a transaction rollback. The following notations are used here: a white circle represents a normal log record, and a shaded circle a CLR; a double-lined arrow from a log record indicates PreviousLSN — its previous log record belonging to the same transaction — and a dotted arrow the UndoNextLSN of the CLR; characters written in a circle identify the executed update operation; an odd number of quotation marks beside the characters represent an undo operation, and an even number of quotes a redo operation. In Figure 5,

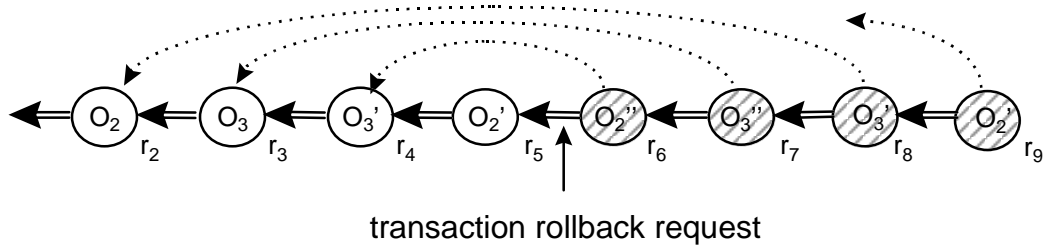


Figure 5: A transaction rollback using the normal log record type.

the operations O_2 and O_3 were already undone when the rollback of the transaction is requested. However, the rollback process redoes the operations O_2 and O_3 and logs these actions in r_6 and r_7 . And then, it undoes the same operations and logs these actions in r_8 and r_9 . Therefore, it unnecessarily performs redoing and undoing of O_2 and O_3 . That is, when using normal log records, there is overhead that all the repetitive undo and redo operations in forward processing are to be undone in the reverse order during a transaction rollback. This is because the rollback process does not distinguish the operations already undone.

We define a new type of log record, called a *partial log record (PLR)*, for efficient support of user-interactive undo. The PLR does not keep their own undo and redo information, but keeps a pointer to reference another log record that contains this information. We call the referenced one an *original log record* since it is a normal log record that corresponds to the original update operation. In the proposed method, the PLR contains only the minimal information to undo and redo; when more information is needed it is read from the corresponding original log record. Therefore, PLRs minimize the log space overhead for undo and redo operations. In addition, a new transaction rollback algorithm using PLRs minimizes the transaction rollback overhead.

In a multi-user environment, it is essential to support logical undo for achieving high concurrency as described in Section 4.1. The PLR incurs two problems in supporting logical undo. First, undos and redos after a logical undo should always perform logical undo in order to find the page that holds the moved data since the recorded PLR for the first logical undo has no undo and redo information. This is very inefficient since logical undo requires additional accesses to the meta data. Second, during the redo pass of crash recovery, logical redo should be performed for the PLR that was recorded for a logical undo during forward processing since the PLR has no redo information. Supporting logical redo is undesirable since it requires significant changes to ARIES that supports only page-oriented redo.

To solve these problems we define another new type of log record, called a *substitute log record (SLR)*, that contains its own undo and redo information for a logical undo. When a logical undo occurs, an SLR is logged instead of a PLR, and its undo and redo information substitutes that of the original log record. Once an SLR is recorded, the following PLRs for undos and redos of the same operation reference this SLR, not the corresponding original log record. When these following PLRs are to be undone, page-oriented undos are

Type
PreviousLSN
UndoNextLSN
OriginalLSN

(a) The structure of a PLR.

Type
PreviousLSN
UndoNextLSN
UndoRedoData

(b) The structure of an SLR.

Figure 6: The structures of new log records.

performed using the undo and redo information of the SLR. Therefore, SLRs minimize the number of logical undos while they avoid duplicating the same undo and redo information. Also, it is always possible to perform page-oriented redo for PLRs using the redo and undo information of the corresponding original log record or SLR during the redo pass of crash recovery. Therefore, SLRs solve both of the problems mentioned above.

Figure 6(a) shows the structure of a PLR. The OriginalLSN field contains the LSN of the corresponding original log record or SLR. The UndoNextLSN field contains the LSN of the next log record to be processed during transaction rollback or bulk undo. The Type field indicates both whether this is a PLR or not and whether the recorded operation is an undo or a redo operation. The PreviousLSN field contains the LSN of the previous log record belonging to the same transaction. Figure 6(b) shows the structure of an SLR. An SLR contains an UndoRedoData field instead of an OriginalLSN field. The UndoRedoData field contains the undo and redo information for a logical undo. The Type field indicates both whether this is an SLR or not and whether the recorded operation is an undo or a redo operation. The PreviousLSN field and UndoNextLSN field are the same as those of a PLR. The next section presents recovery algorithms using PLRs and SLRs.

4.2.2 Recovery Algorithms

In this section, we present new recovery algorithms related to the user-interactive undo facility. First, we describe user-interactive undo in forward processing of a transaction and present an algorithm for rollback of transactions containing user-interactive undo. Next, we show that the crash recovery algorithm of ARIES can be used in our method without significant changes. Finally, we describe a bulk undo as a new operation for undoing with one interaction a sequence of operations performed during forward processing of a transaction.

User-interactive undo in forward processing of a transaction During forward processing of a transaction, user-interactive undo is applied in the reverse order starting from the last executed operation. User-interactive undo in forward processing can be classified into three types of operations: undo of an original update operation, undo of

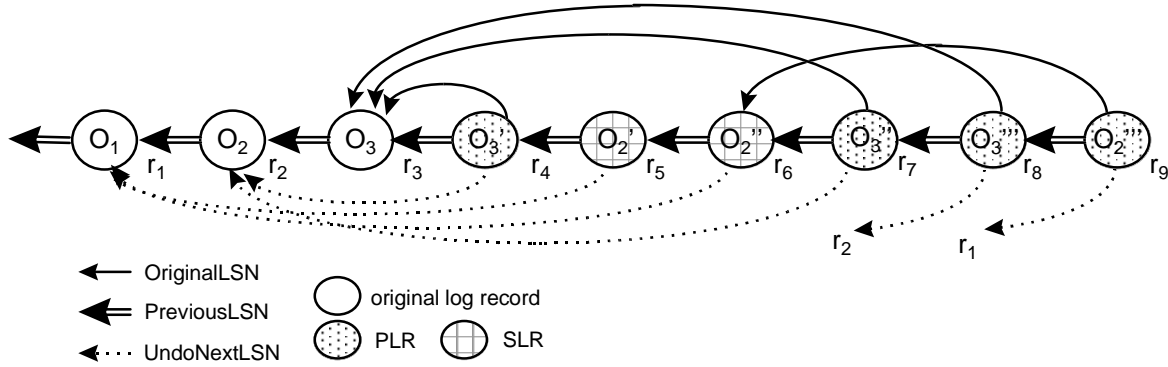


Figure 7: An example of user-interactive undo logged by PLRs and SLRs.

an undone operation, and undo of a redone operation. The undo operation of an original update operation undoes the corresponding original log record. This operation uses undo information of the original log record and logs a PLR with the following values. The Type is undo PLR; the OriginalLSN is the LSN of the original log record that has just been undone; the PreviousLSN is the LSN of the log record that has most recently been logged by the same transaction; the UndoNextLSN is the PreviousLSN of the original log record. This value of UndoNextLSN indicates that the previous log record of the original log record, but not the previous log record of the PLR, is the next log record to be rolled back during a transaction rollback. When a logical undo occurs, an SLR is recorded instead of a PLR. The SLR's Type is undo SLR, and its UndoRedoData is undo and redo information for the logical undo. Its PreviousLSN and UndoNextLSN are the same as those of a PLR.

Figure 7 shows an example of user-interactive undo logged by PLRs. White circles represent normal log records, dot-filled circles PLRs, and grid-patterned circles SLRs. Single-lined arrows indicate OriginalLSNs pointing to the corresponding original log records of the PLRs, and dotted arrows UndoNextLSNs pointing to the previous log records of the original log records, and double-lined arrows PreviousLSNs. In Figure 7, there are two continuous undo operations logged by r_4 and r_5 . The record r_4 is an undo PLR that records a page-oriented undo of r_3 ; its original log record, next log record to be undone, and previous log record are r_3 , r_2 , and r_3 , respectively. The record r_5 is an undo SLR that records a logical undo of r_2 ; its next log record to be undone and previous log record are r_1 and r_4 , respectively.

An undo operation of an undone operation is the same as a redo operation of the original operation. The undo of a PLR logged for an undone operation is performed using the redo information in the original log record or the undo(or redo) information in the undo(or redo) SLR. This action is also logged by a PLR with the following values. The Type is redo PLR, the OriginalLSN and the UndoNextLSN are the same LSNs as those in the PLR that has been undone. The undo of an SLR logged for an undone operation is performed using its own undo information. This action is logged by a PLR with the following values. Its UndoNextLSN is the same as that of the SLR, and its OriginalLSN is the LSN of the SLR. When a logical undo occurs, an SLR is recorded instead of a PLR.

The SLR's Type is redo SLR and its UndoRedoData is undo and redo information for the logical undo. Its PreviousLSN and UndoNextLSN are the same as those of the PLR. In Figure 7, there are two continuous redo operations logged by r_6 and r_7 . The record r_6 is a redo SLR that records the logical undo of undo SLR r_5 . Therefore, r_6 points to r_1 as its next log record to be undone. The record r_7 is a redo PLR that logs undoing the undone operation logged by undo PLR r_4 . Therefore, r_7 and r_4 point to the same original log record, r_3 , and the next log record to be undone, r_2 .

An undo operation of a redone operation is the same as an undo operation of the original operation. Except that the type of operation is undo, this operation is processed in almost the same way as in the redo operation that has just been described. The undo of a PLR logged for a redone operation is performed using the undo information in the original log record or the redo(or undo) information in the undo(or redo) SLR. This undo action is logged by a PLR with the following values. The Type is undo, the OriginalLSN and the UndoNextLSN are the same LSNs as those in the PLR that has been redone. The undo of an SLR logged for a redone operation is performed using its own undo information. This action is logged by a PLR with the following values. Its UndoNextLSN is the same as that of the SLR, and its OriginalLSN is the LSN of the SLR. In Figure 7, there are two continuous undo operations logged by r_8 and r_9 . The record r_8 is an undo PLR that logs undoing the redone operation logged by undo PLR r_7 . Therefore, r_8 and r_7 point to the same original log record, r_3 , and the next log record to be undone, r_2 . The record r_9 is an undo PLR that records undoing the redone operation logged by redo SLR r_6 . Therefore, r_9 points to r_6 as its original log record, and r_1 as its next log record to be undone.

A transaction rollback algorithm The history undo model has the problem that the transaction rollback takes much more time than is necessary in case the transaction contains repetitive undone and redone operations. For resolving this problem, we propose a new transaction rollback algorithm that avoids executing unnecessary operations by skipping these repetitive undo and redo operations using the Type and UndoNextLSN fields of PLRs and SLRs. The algorithm is as follows. Let $R_{current}$ be the last log record of a transaction to be rolled back.

Algorithm Transaction Rollback:

1. Read $R_{current}$.
2. If $R_{current}$ is a normal log record, do the corresponding undo action and write a CLR for this action; set the UndoNextLSN of the CLR to the PreviousLSN of $R_{current}$; set $R_{current}$, a log record to be accessed next, to the PreviousLSN of $R_{current}$.
3. If $R_{current}$ is a CLR, set $R_{current}$ to the UndoNextLSN of $R_{current}$.
4. If $R_{current}$ is a PLR or an SLR, do the following actions:
 - (a) if the Type of $R_{current}$ is undo, go to Step 4(c);
 - (b) if the Type of $R_{current}$ is redo, do the following actions:
 - i. if $R_{current}$ is a PLR, read $R_{original}$ — the log record referenced by the OriginalLSN of $R_{current}$;

- ii. do the corresponding undo action and write a CLR for this action; set the UndoNextLSN of the CLR to the UndoNextLSN of $R_{current}$;
 - (c) set $R_{current}$ to the UndoNextLSN of $R_{current}$.
5. go to Step 1.

During transaction rollback, SLRs can be handled in almost the same way as PLRs. It is because SLRs are recorded instead of PLRs when logical undos occur, and SLRs contain the same kind of information as PLRs except that SLRs contain undo information instead of OriginalLSN. This algorithm handles SLRs in the same way as PLRs except that undoing SLRs skips Step 4(b)i— a step to access the original log records. For ease of explanation, we regard SLRs as PLRs hereafter.

This algorithm has two skipping processes, which skip the log records already undone in transactions. First, in Step 4(a), when the Type of $R_{current}$ is undo, the algorithm skips undoing it, since the original operation has already been undone before the rollback. This is similar to the idea used in ARIES that never undoes CLR. Second, in Step 4(c), it skips all the log records between the PLR and the corresponding original log record of the PLR since the next log record to be undone is the UndoNextLSN instead of the PreviousLSN of $R_{current}$. That is, if there are n undo operations before a transaction rollback request, the LSN of the next log record to be accessed after undoing the log record for the n -th undo is the PreviousLSN of the corresponding original log record, not the PreviousLSN of the current log record. This means it skips at least $(n - 1)$ PLRs between the PLR and the original log record. Though, unlike in ARIES, this skipping area includes both undo and redo PLRs, we can still use the UndoNextLSN concept since we follow the history undo model. In this model, if the n -th undo operation is redo, the state reached by this operation is the same as the one reached by the original operation. Therefore, undoing the n -th operation means undoing the original operation. These skipping processes, which skip the repetitive undo and redo operations, reduce the number of log records to be processed, and therefore, enable faster rollback of transactions.

Figure 8 shows an example of a transaction rolled back by the proposed algorithm. At the requesting point of the transaction rollback, the operation O_2 was undone at r_5 and redone at r_6 , and the operation O_3 was undone at r_4 . The rollback process has undone only two operations instead of six; it has undone the operation O_2 that was redone at r_6 and the operation O_1 at r_1 , but has skipped the log records r_5 , r_4 , r_3 , and r_2 . In this way the proposed algorithm skips undoing repetitive undo and redo operations and undoes only the operations still reflected in the database.

Crash recovery algorithms We utilize crash recovery algorithms of ARIES, which consist of the analysis pass, redo pass and undo pass. User-interactive undo has no effect on the analysis pass of determining the starting point of the redo pass and loser transactions. During the redo pass of repeating history, we have to perform additional operations that read the redo information from the original log records when redoing PLRs. During the undo pass, we use the same transaction rollback algorithm as mentioned above.

Bulk undo A *bulk undo* is defined as a new operation that undoes a sequence of user operations with just one interaction. The bulk-undone operations can be undone like an un-

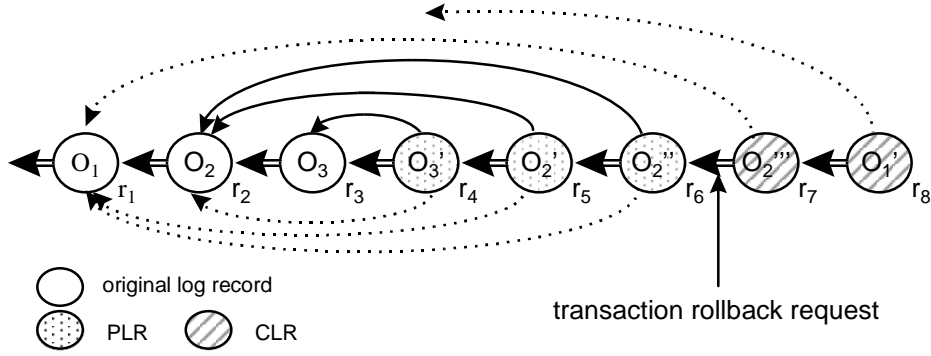


Figure 8: An example of a transaction rollback.

done operation. Before requesting bulk undo operations, undopoints should be established during forward processing of a transaction to mark the states to be restored. When a user requests establishing an undopoint at any state, the LSN of the latest log record written by the transaction, called *UndopointLSN*, is stored in virtual storage and the corresponding undopoint identifier is returned. A bulk undo to an undopoint undoes all the log records from the last to the undopoint and writes PLRs for these actions. The PLRs logged by a bulk undo keep additional information of the undopoint identifier to differentiate them from other PLRs for ordinary undo operations.

The bulk undo to an undopoint resembles partial rollback to a savepoint, but differs in that it allows undoing bulk-undone operations. This is possible since we do not release the locks acquired after the undopoint and log the bulk undo operation using the undoable PLRs.

During the bulk undo, like in a transaction rollback algorithm, there are skipping processes, which skip the log records already undone as well as repetitive undo and redo operations. After a sequence of operations is bulk-undone, these operations can be redone by one undo operation. This example is shown in Figure 9. Dark circles represent PLRs written for the operations that were undone or redone by bulk undo operations. *Undopoint₁* was established after executing the operation O_1 . At t_1 , a bulk undo to *Undopoint₁* was requested. This operation undid two operations, O_4 and O_3 , but skipped r_3 and r_2 written for the operation O_2 since it had already been undone. At t_2 , after executing the operation O_5 , two undo operations were requested. The first one undid log record r_8 and wrote this action at r_9 ; the second one undid a bulk undone operation from r_7 to r_6 and wrote these actions at r_{10} and r_{11} .

Figure 10 shows another example of bulk undo operations with two undopoints. *Undopoint₁* was established after executing the operation O_1 and *Undopoint₂* after O_3 . At t_1 , a bulk undo to *Undopoint₁* was requested. This operation undid two operations, O_4 and O_3 , but skipped r_3 and r_2 written for the operation O_2 since it had already been undone. At t_2 , after executing the operation O_5 , a bulk undo to *Undopoint₂* was requested. This operation undid log records from r_8 to r_5 : it undid operation O_5 , redid operation O_3 , and wrote these actions at r_9 and r_{10} , but skipped r_6 and r_5 written for the operation O_4 since it

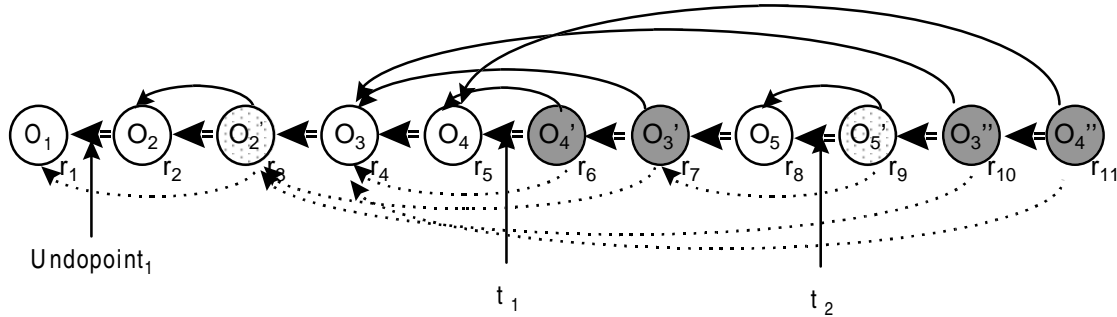


Figure 9: An example of a bulk undo operation and its redo.

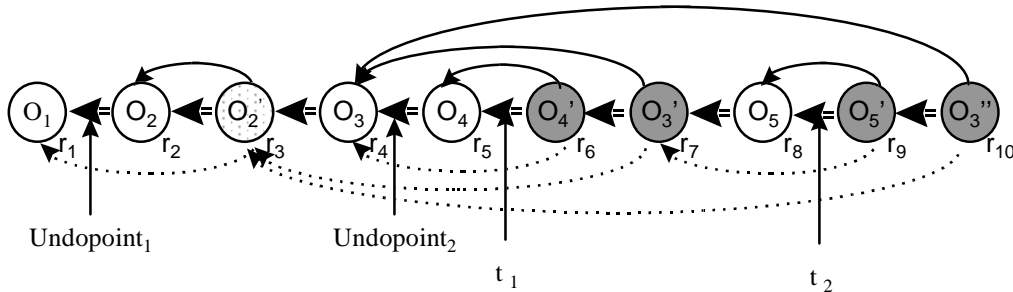


Figure 10: An example of bulk undo operations with two undopoints.

had been executed and undone after *Undopoint₂*. In ARIES, this scenario is not possible since after the first partial rollback to *Undopoint₁*, *Undopoint₂* is no more outstanding as described in Section 4.1.

The bulk undo is useful for both the users and the system. It is convenient for the users since they can undo a sequence of operations with just one simple interaction. From the system's viewpoint, the bulk undo is efficient since it reduces interactions between the user and the system; it also reduces the number of log records to be processed during bulk undo by skipping repetitive undo and redo operations.

4.3 Performance Analysis

As described in the previous section, the proposed method can be implemented by extending ARIES. This section analyzes the overhead of extending to support user-interactive undo by comparing our method with ARIES. This comparison is restricted to undo operations since redo operations are not provided by the partial rollback facility in ARIES and cannot be compared. We also restricted this comparison to page-oriented undo only since the probability that logical undo occurs due to uncommitted updates of one transaction being moved to a different page by another transaction is relatively low. We treat a partial

rollback as a bulk undo operation in the proposed method.

Generally, it is very complex to analyze the performance of recovery method since one has to take into account a number of aspects. The most important measures as described by Reuter[13] are as follows: 1) overhead during normal database processing; 2) recovery speed after a failure; 3) space requirement of the log file. The overhead during normal database processing and speed of crash recovery can be analyzed by the log access time, data access time, and execution time. If the same transaction were executed, our method and ARIES would require the same data access time and execution time, but different log access time since they use different log record types for undo operations. The log access time is proportional to the number of log records to be read, the number of log records to be written, and the size of a log record. We use these factors to analyze the overhead during normal database processing and recovery speed after a failure.

During normal database processing, transactions may be in forward or rollback processing. In forward processing, although the proposed method writes PLRs for undo operations while ARIES writes CLRs, the numbers of log records read and written by the two methods are identical. In rollback processing, the two methods access the same number of log records since both of them skip the log records already undone, undo other log records, and log these undo actions. Therefore, during normal database processing, the proposed method accesses the same number of log records as ARIES.

Restart recovery consists of the analysis pass, redo pass, and undo pass. During the analysis pass, two methods read the same number of log records since they have recorded the same number of log records during the normal database processing. However, during the redo pass, the proposed method has to read the original log records of PLRs to get the redo information in case their updates have not yet been reflected on the database. This is because PLRs do not keep the redo information to avoid the replication of the same information. In the worst case, the overhead is significant since reading the original log record requires reading another log page; however, this overhead may be reduced when the original log record is stored in the same log page containing the PLR and when the original log records for a sequence of PLRs are stored in the same log page. During the undo pass, the number of log records to be read and written are identical in the two methods since the undo pass is similar to transaction rollback.

The size of a PLR is quite smaller than that of a CLR. This is because a CLR keeps the redo information while a PLR only keeps the LSN of the corresponding original log record. Redo information in a CLR is quite larger in size than an LSN and can be very large in some cases depending on the types of operations that have been logged.

Compared with ARIES, the proposed method performs the normal database processing as fast or even slightly faster since it accesses the same number of log records whose sizes are smaller than CLRs of ARIES. Similarly, our method performs the analysis pass and undo pass during crash recovery as fast as or even faster than ARIES. During the redo pass, however, if the database already reflects the PLRs' updates, our method works faster than ARIES; otherwise, it has additional overhead of reading the corresponding original log records. The overall performance of crash recovery is comparable to that of ARIES in spite of the additional overhead in the redo pass since it is as fast as or even faster in the analysis and undo passes.

The space requirement of the log file can be expressed by the number of log records

written times the size of a log record. Since both methods have the same number of log records to be written and the size of a PLR is shorter than a CLR, the space requirement of the log file in our method is slightly smaller than in ARIES.

In summary, the proposed method provides performance comparable to that of ARIES in normal database processing, crash recovery, and space requirement. This result indicates that our method supports the user-interactive undo without performance degradation compared with existing recovery facilities, especially those in ARIES.

5 Conclusions

During authoring processes in new DBMS applications such as software development, hypermedia authoring, and CAD, there are frequent interactions between users and the system. Therefore, the applications should support such facilities that make these interactions easily. In particular, in authoring processes, users need user-interactive undo for correcting their trials and errors easily by undo and redo operations.

Previous research on user-interactive undo has been restricted on its models and user-interactive undo has been provided by the application programs themselves. Since the implementation of user-interactive undo is quite complex, it poses significant burden to application programmers. Moreover, if an application employs a DBMS, it has additional overhead for handling the data the DBMS itself updates.

In this paper, we propose a new recovery method with which a DBMS can support user-interactive undo. The proposed method can be implemented by extending ARIES, a well-known recovery method. In particular, we support the user-interactive undo without altering the basic idea of a crash recovery facility in ARIES. We adopt the history undo model as our undo model because it has a nice property of being able to roll back to any previous state. We have proposed the notion of the PLR and SLR that allows redo of undone operations and resolves the repetition problem of the history undo model. The primary role of the SLR is to avoid logical redo and, at the same time, to reduce the number of logical undos.

The proposed method provides both user-controlled undo and redo operations in a DBMS while the partial rollback facility in ARIES provides only undo operations. In addition, its performance is comparable to that of ARIES. In this paper, we present two solutions to the problem in the history undo model: the first is an algorithm that guarantees fast rollback of a transaction in spite of repetitive undo and redo operations; the second is a bulk undo operation by which a user can restore the database to a predetermined state with one interaction. Providing user-interactive undo in the DBMS is a new concept, and we expect that our new DBMS facilities relieve application programmers of the overhead of implementing user-interactive undo themselves in DBMS applications.

References

- [1] Archer, J.E., Conway, R., and Schneider, F.B., "User Recovery and Reversal in Interactive Systems," *ACM Trans. on Program. Lang. Sys.*, Vol. 6, No. 1, pp. 1–19, Jan. 1984.

- [2] Berlage, T., "A Selective Undo Mechanism for Graphical User Interface Based on Command Objects," *ACM Trans. on Computer-Human Interaction*, Vol. 1, No. 3, pp. 269–294, Sept. 1994.
- [3] Bernstein, P.A., Hadzilacos, V., and Goodman, N., *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.
- [4] Ginige, A., Lowe, D.B., and Robertson, J., "Hypermedia Authoring," *IEEE Multimedia*, pp. 24–35, 1995.
- [5] Gray, J. et al., "The Recovery Manager of the System R Database Manager," *ACM Computing Surveys*, Vol. 13, No. 2, pp. 223–242, June 1981.
- [6] Gray, J. and Reuter, A., *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993.
- [7] Haerder, T. and Reuter, A., "Principles of Transaction-Oriented Database Recovery," *ACM Computing Surveys*, Vol. 15, No. 4, pp. 287–317, Dec. 1983.
- [8] Kim, W., *Modern Database Systems*, The ACM Press, 1st Edition, 1995.
- [9] Korth, H. and Speegle, G., "Long-Duration Transactions in Software Design Projects," In *Proc. of the Sixth Intl. Conf. on Data Engineering*, IEEE, pp. 568–574, Feb. 1990.
- [10] Korth, H., Kim, W., and Bancilhon, F., "On Long-Duration CAD Transactions," *Information Sciences*, Vol. 46, No. 1-2, pp. 73–108, 1988.
- [11] Mohan, C. et al., "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollback Using Write Ahead Logging," *ACM Trans. on Database Systems*, Vol. 17, No. 1, pp. 94–162, Mar. 1992.
- [12] Mohan, C. and Levine, F., "ARIES/IM: An Effective and High Concurrency Index Management Method Using Write-Ahead Logging," In *Proc. Intl. Conf. on Management of Data*, pp. 371–380, ACM SIGMOD, 1992.
- [13] Reuter, A., "Performance Analysis of Recovery Techniques," *ACM Trans. on Database Systems*, Vol. 9, No. 4, pp. 526–559, Dec. 1984.
- [14] Prakash, A. and Knister, M.J., "A Framework for Undoing Actions in Collaborative Systems," *ACM Trans. on Computer-Human Interaction*, Vol. 1, No. 4, pp. 295–330, Dec. 1994.
- [15] Thimbleby, H., *User Interface Design*, Addison-Wesley, 1990.
- [16] Wright, P., "Designing the Human-Computer Interface to Hypermedia Applications," In Book *Designing Hypermedia for Learning*, (Jonassen, D.H. and Mandl, H. eds.), Springer-Verlag, 1990.