

Recovery in Distributed Extended Long-lived Transaction Models*

M. M. Gore[†] and R. K. Ghosh

Department of Computer Science and Engineering,
Indian Institute of Technology, Kanpur, India
gore, rkg@iitk.ernet.in

Abstract

This paper addresses the recovery and the rollback problem in distributed collaborative transactions. We propose a solution to the problem in a generalized ARIES [9] framework. We modified its existing data structures and provided additional data structures for recovery of distributed extended long-lived transactions. In the proposed model the transactions communicate and collaborate only by exchanging messages. The messages are logged along with usual database actions. In recovery of distributed extended transactions these message logs and message tables are extensively used. The recovery algorithms presented in this article, work in distributed environment, under extended transaction models, with different kind of failures and transaction rollback.

1. Introduction

The ability to recover from failures and erroneous executions is an essential requirement for database systems. This article focuses on the recovery manager and its interactions with other components of the *database management system* (DBMS). The recovery management in database systems provides engineering solutions to various kind of failures, offering reliability by restoring the system to a coherent and consistent state. Thus the major goal of the recovery manager is to ensure the atomicity and durability of transactions. The complexity of recovery mechanism is further enhanced in case of extended transaction model in a fully distributed environment. In this paper we propose a solution for the above problem with ARIES concepts augmented by message logging mechanism.

Due to the importance of recovery management in DBMS, many researchers have focussed attention on the design and implementation of recovery algorithms. Haerder

and Reuter [7], provides a useful taxonomy of recovery techniques and their respective characteristics. These concepts are amplified in further details by Bernstein, Hadzilacos, and Goodman [1] and by Gray and Reuter [6]. Mohan et al. [9] proposed ARIES (Algorithm for Recovery and Isolation exploiting Semantics), which supports partial roll-backs of transactions, fine granularity locking and recovery using write-ahead logging. ARIES makes use of repeating history paradigm to redo all missing updates before performing the rollback of the loser transactions during restart after system failure. There are different variations of ARIES such as ARIES/NT [14] that supports nested transaction presented by Rothermel and Mohan. Another variation is ARIES/CSA, proposed by Mohan and Narang [10] for client server architecture. Panagos and Biliris [11] presented a scheme for client server architecture which is based on write ahead redo logging only. Log records are generated at the clients and shipped to the server during normal execution and transaction commit. Transaction roll-backs are fast because there are no updates that have to be undone. Rajeev Rastogi et al. [13] presented recovery schemes for distributed main-memory databases, specifically for client server and shared disk architecture. They also employ shipping of log records to the server in client server architecture. In shared disk they present two alternatives either page shipping or the broadcasting of the log of updates.

There are large number of extended transaction models that have been proposed but their scope and utility is limited to the applications they model. The above scenario has given users very little scope of flexibility. There are attempts to provide flexible extended transaction facility e.g. ASSET [2] and DiET [12]. DiET has implemented distributed extended transaction models but has not provided any extended transaction recovery. In this paper we present uniform recovery algorithms for different models of distributed extended transactions. This work will help to build a uniform recoverable, distributed, extended, long-lived transaction framework for different kind of advanced applications.

The outline of this paper is as follows. Section 2 presents

*Partially supported by project number AICTE/CSE/96264

[†]On leave from M.N.R.Engg. College Allahabad under QIP

brief description about various definitions, primitives for modeling extended transactions and transaction dependencies, system model and related assumptions and detailed description of data structures used in recovery. Section 3 details recovery algorithms and related topics. Case studies are analyzed and discussed in section 4.

2. Definitions and Data Structures

This section presents background definitions used in database recovery. The primitives used in modeling collaborative transactions, and commonly found dependencies among collaborative transactions are presented next. Finally, a description of system model and major data structures used in the recovery algorithms is given.

The *recovery log* is a sequentially written structure on the persistent storage containing information required for the purpose of recovery. A *before-image* of a data item X is a copy of the most recently *committed* value of X stored in the log. An *after-image* of a data item X is a value of X written by an active transaction stored in the log. The process of writing a before-image of a data item to the log before allowing the data item to be overwritten is known as *write-ahead logging (WAL)*. A before-image helps in undoing the effects of a losing transaction during recovery while after-image helps in restoring correct value in database written by a committed transaction.

Check-pointing is the process of ensuring that all logged information including dirty page table and transaction table are stored in stable storage. The three types of check-pointing strategies are *commit consistent*, *action/cache consistent* and *fuzzy*. A recent survey on these issues can be found from [8].

Extended transaction models [5] are defined using certain transaction primitives. Normally an user is not required to use these primitives. But these will be used in the code generated by a compiler for a database programming language or an application development interface, that provides high-level support for extended transactions. For more details the readers may refer Chrysanthis and Ramamritham [3]. Three important primitives which are designed mainly to support collaborative distributed transaction models are as follows.

- **ResponsibleTr** $(p_{t_i}[ob])$: It identifies the transaction responsible for committing or aborting the operation $p_{t_i}[ob]$ with respect to the stable log. $p_i[ob]$ denotes the invocation of operation p on object ob by transaction t . Generally the event invoker transaction is held responsible unless it passes its responsibilities to other transaction through delegation.
- **Delegate** $_{t_i}[t_j, p_{t_i}[ob]]$: It denotes that a transaction t_i delegates the responsibility for committing or aborting

the operation $p_{t_i}[ob]$ to another transaction t_j . Delegation cannot occur in the event that the delegatee has already committed or aborted and/or is not in a position to own the responsibility. Delegation redirects the dependencies induced by the delegated operations from the delegator to the delegatee.

- **Permit** $_{t_i}[t_j, p_{t_i}[ob]]$: It means a transaction t_i permits another transaction t_j to view the operation $p_{t_i}[ob]$.

Many types of dependencies possible between transactions. Following is a non exhaustive list of dependencies occurring too often in the case of extended transaction models.

- **Commit Dependency**: For two transactions t_i and t_j if both commit t_j cannot commit before t_i commits, but if t_i aborts t_j still may commit.
- **Abort Dependency**: For two transactions t_i and t_j if t_i aborts, t_j must abort.
- **Group Commit Dependency**: For two transactions t_i and t_j either both commit or neither commits.

The interactions among the transactions are triggered through messages. The responsibility of delivering the message lies solely with the communication manager at the site. Besides message data, messages also carry message identifier, issuing transaction id, receiving transaction id, message type and PrevLSN for the transaction which has sent the message. In normal processing messages are logged only when they are put on the stable storage. The advantage of this scheme is that transaction log reflects the message delivery while communication manager can execute repeated tries for successful delivery of the message. Similarly the communication manager at receiving site puts the received message on the stable storage and then delivers it to transaction it is intended for. The receiving transaction also logs the action of message receipt. The communication manager ensures that no message is delivered to a transaction twice.

The **system model** is as follows.

- There are N sites. Each site has a Transaction Manager, a Lock Manager, a Storage Manager, a Recovery Manager, and a Communication Manager. All sites possess stable storage. The database is distributed among these sites. The logging is performed at all sites.
- Transactions execute at one or more sites. For distributed transactions two-phase commit protocol is used. Each site maintains its own portion of the log. Transactions can be simultaneously distributed and/or extended. Extended transaction can be distributed but

not in strict distributed transaction case. For example in Join transaction model, transaction t_a at site X joins transaction t_b at site Y. So, Join is distributed to two sites but transaction t_b need not to undergo 2PC protocol.

- Recovery managers co-ordinate among themselves for distributed recovery, though it is possible that only a single site has failed.
- The network is FIFO and reliable.

Transaction system is assumed to be asynchronous. There exists no bound on relative speeds of nodes and thereby transactions residing on the nodes. There is also no bounds on message transmission delays. We model recovery for the extended transaction models which are assumed to be long-lived and generally of collaborative nature. We assume the interactions among the transactions are through messages and messages are not too frequent to burden the system. For example in nested transaction model, usually a parent will send a message to each of its children and in turn it receives one message from each of its children. We assume further that the interactions among the transactions are known a priori. Therefore the message send receive pattern from one transaction to other and also message placeholders in the transaction program are known.

2.1. Data Structures

Some modifications in the ARIES data structures were necessary to make them more general and suitable for the distributed extended transaction model. Alongside we also added message tables (one each for messages send and messages received) with existing data structures log, transaction table, dirty page table and the pages that contain the data item.

The Log

The log knows everything as it maintains complete history. It is a sequence of log records. Each log record contains some housekeeping information but major part of it contains redo/undo information. In the Figure 1 details of individual log record are given. One important point to be noted here is that of inclusion of ResponsibleTrID. This is a pointer to the transaction responsible for the actions logged in this log record. This simplifies understanding of extended transaction scenario where responsibilities are transferred from one transaction to other and this will also hold true for the traditional transaction environment where no responsibility transfer takes place. Thus responsibility transfer is also reflected in the log justifying the universal assumption regarding log that it knows everything.

LSN helps in distinguishing one log record from other. In the recovery and roll back two LSNs are very important. they are

```

LogRec:      Record
LSN:        Log sequence number;
Type:       {Update, Comp, Mesg, BgnChkpt,
             EndChkpt, OsFileReturn, End };
ResponsibleTrID: Transaction identifier;
             /* compare with ARIES */
PrevLSN:    Log sequence number;
UndoNxtLSN: Record
Page ID:    Page Identifier;
Data:       Case (Type) of

             Update or Comp: Record
             DataItemName: Name of the data item;
             BeforeImage:  Value;
             AfterImage:   Value;
             endrecord;

             Mesg:         Record
             MessID:       Message Identifier;
             sentTrID:     Transaction identifier;
             recdTrID:     Transaction identifier;
             Type:         { Sent, Receive };
             endrecord;

             End:
             Status:       {Commit, Rollback};

             EndChkpt:    Record
             TrTable:     Transaction Table;
             DirtyPages:  Dirty Pages Table;
             endrecord;

endcase;
endrecord;

```

Figure 1. The log record

- SaveLSN: This is used as low water mark for partial roll back of a transaction. A *SaveLSN* = 0 means the corresponding transaction is to be rolled back completely.
- MasterRec: This is the LSN of latest completed checkpoints begin checkpoint log record before the crash. From this LSN the history is repeated with the help of log records and transaction tables and dirty page tables are also reconstructed.

The Transaction Table

```

TrTableEntry: Record
TrID:         Transaction identifier;
LastLSN:     Log sequence number;
UndoNxtLSN:  Log sequence number;
LastSavePoint: Log sequence number;
Status:      { 'U', 'P', 'C', 'A' }
             /* Active,Prepared,Committed,Aborted*/
endrecord;

```

Figure 2. The transaction table entry

The entries of a transaction table are shown in Figure 2. This resides in the volatile memory and its contents are copied to stable storage during a checkpoint and get reflected in the *end checkpoint log record*. Since in the extended long-lived transactions, complete roll back in the event of crash means huge loss of work, save points are a necessity. One field entry in a transaction table record keeps track of the last save point. These are generally the message send or receive points in the event history of a transaction.

The Dirty Page Table

The dirty page table has two entries one is *PageID*, the page identifier while other is *RecLSN*. The *RecLSN* is used in the redo pass after a crash has taken place. In the dirty pages table the minimum of *RecLSN* provides the pointer to the log from where the updates have not being reflected to non volatile storage.

The page

The page contains PageLSN field besides the PageID. The PageLSN reflects the latest update made to this page as it is LSN of Log of the said update action.

The Message Tables

The entries in a message table are shown in Figure 3. The record entries in the message tables contain all the information which has passed from one transaction to other. In the extended transaction environment the responsibility transfer (*Delegate*) or controlled relaxation of ACID properties (*Permit*) is done only through messages. Both sender and receiver of control transfer must have the complete information in the stable storage for proper execution of control transfer to take place in committing, aborting and recovering an extended transaction after a crash. As log contains everything we transfer the list of LSN from one transaction to other. This transfer is logged and permissions are changed accordingly. In the case of *delegate* the ResponsibleTrID would point to delegatee after the transfer while previously it was pointing to delegator.

```

MessageTableEntry:   Record
MessID:              Message Identifier;
sentTrID:            Transaction identifier;
recvTrID:            Transaction identifier;
PrevLSN:             Log Sequence Number;
controlype:          {Delegate, Permit, Ack};
LSNList:             List of LSN
                    /* delegated or permitted */
endrecord;
```

Figure 3. An entry in the message table

As notified earlier we put message entries in the stable storage, then we notify communication manager for message delivery and log the action. Similarly for the incoming messages the communication manager puts message in

stable storage and then notifies the concerned transaction which then logs the receive message action. Thus at each site there are two messages tables one for the outgoing messages and other for incoming messages. The argument here for putting the message entries on the stable storage are that they are invariant unlike transaction table or dirty page table entries which change as transactions make progress. In the check-pointing process we do take transaction table and dirty page table entries to the stable storage and in the event of crash we build the transaction table and dirty page table through checkpoint record. For message tables both of these actions are not required. Further communication manager will have a track of messages it delivered and received in the history.

3. Algorithms

This section describes how messages are used in the normal processing and the role played by message tables in the event of partial and total rollback, and the different phases of recovery after a crash has taken place.

3.1. Normal Processing

In the normal processing the locking, latching and concurrency control is similar to ARIES. For Distributed Extended Transaction model we make use of two primitives Delegate and Permit which allow us to model extended transaction models. Now let us consider what happens when a transaction sends a list of LSN's to other transaction with control message Delegate or Permit. We deal these cases separately.

• Delegate

When a transaction delegates its operations to another transaction, the latter becomes responsible for committing or aborting these operations of the former on database objects. In this case the log of delegator transaction should reflect the responsible transaction as the delegatee and not the delegator transaction for these delegated operations. Further as the log is transferred all the corresponding locks have to be transferred from the delegator to the delegatee to enable it to redo or undo these delegated operations. Thus when a delegate message is sent, lock and log responsibility transfer has to be carried out simultaneously by the recovery manager and the lock manager at the delegator. The ResponsibleTrID is a pointer to identifier of transaction who owns the responsibility of operation logged. Before delegation it was the transaction which carried out the operation while after delegation it will point to delegatee which has assumed the responsibility of committing or aborting the operations this is shown in the Figure 4.

The message to delegatee carries the list of LSN delegated. The operations of a transaction are chained by the

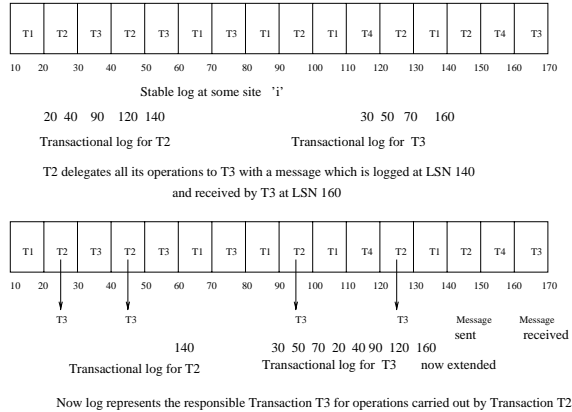


Figure 4. Change of Transactional log when Delegate message takes place

PrevLSN field thus forming a linked list. At the receipt of delegate message, the recovery manager at delegatee site appends on the linked list formed by PrevLSN of delegatee, the list of LSN sent by the delegator. This is pointed out in the extended transaction log of T3 shown in the Figure 4. It is possible in this enhanced logging scheme to transparently delegate a set of operations from one transaction to the other transaction. The case of complete delegation is found in the case of nested transaction when child commits thereby delegating all its operations to its parent.

• **Permit**

In the case of permit the message again carries the List of LSN but now the difference with the delegate case is that in the case of permit the responsible Transaction remains the permitter. The scenario changes as now the locks are held by the permittee while locks are retained by the permitter. Since permittee is allowed to access Log records given by the list of LSN all the operation done on the objects are also visible to the permittee. In permit primitive the permittee is allowed to abort but is disallowed to commit before the permitter thus binding itself in the commit dependency. In the nested transactions case parent permits its child to view operations carried by it before the child has been spawned, but the operations carried out by child are finally committed only when parent commits.

Now we further substantiate the argument of putting the message on the stable storage for control actions delegate and permit both at sending site and receiving site. These messages are flag holders for the transfer of control of operations and locks on the objects in the case of delegate where complete responsibility transfer takes place, and transfer of hold on the locks on objects by permitter takes place. Thus messages are crucial in the control flow of collaborative transaction systems and are automatic and appropriate

choice for save points. In the long-lived transactions one is ill at ease in completely rolling back the active transactions in the event of a crash i.e. losing the work done for days or weeks. The message log points also stand as appropriate choice for the saving lock information and application cursor information for re-executing the transaction from this save point after a crash thus minimizing the damage done by failure.

The Rollback

The rollback of a transaction at a site is managed by the recovery manager of that site. In the distributed extended transaction environment, recovery managers at one or more sites are required to manage correct rollback of an extended transaction.

The rollback algorithm presented here is based on ARIES framework. This algorithm is transaction based and rolls back the transaction up to a given SaveLSN. The rollback algorithm for distributed extended transaction environment is described in Figure 5.

```

Rollback (SaveLSN, TrID)
UndoNxt:= TrTable[TrID].UndoNxtLSN;
WHILE (SaveLSN < UndoNxt) Do
  LogRec:= LogRead ( UndoNxt )
  SELECT ( LogRec.Type )
  WHEN ( Update ) Do
    {undo operation; UndoNxt:= LogRec.PrevLSN};
  WHEN ( Comp ) Do
    {UndoNxt:= LogRec.PrevLSN};
  WHEN ( Mesg ) Do {
    If (LogRec.Data.Type = Receive)
    {UndoNxt:= LogRec.PrevLSN};
    /*For both Delegate and Permit control type */
  }
  Else {
    mid := LogRec.Data.MessID;
    SELECT (SentMessageTable[mid].controltype)
    WHEN (PERMIT) Do {
      tid :=SentMessageTable[mid].rcvTrID;
      Get siteid for this tid;
      Contact Recovery Manager and ask it to
      rollback transaction given by tid, up to
      PrevLSN given in the receive message
      table, in the entry corresponding to mid.
      UndoNxt:= LogRec.PrevLSN;
    }
    Otherwise UndoNxt:= LogRec.PrevLSN;
  }
End SELECT /* Message Control Type */
}
Otherwise UndoNxt:= LogRec.PrevLSN;
End SELECT /* Log Rec Type */
End WHILE

```

Figure 5. Distributed Rollback Actions

As shown in the Figure 5 we follow the same notions as given in ARIES except for the case when LogRec Type is message. The message is either received or sent out by the transaction. The message control type is either permit or

delegate. All the four possible cases are described below.

- Message received and control type is permit, no recovery action is necessary. This is due to the fact that the message receive point precedes all operations after permit. So all such operations have already been undone.
- Message received and control type is delegate, the PrevLSN of delegatee would be the last LSN of delegated list of LSN. The first member of delegated list of LSN will point to the PrevLSN of delegatee's log before the message receive point. Hence the rollback is managed transparently by recovery managers at the delegatee transaction site and delegator transaction site.
- Message sent and control type is permit, permitter transaction has to release permitted locks while rolling back. Thus a cascading effect of forced rollback on the permittee transaction takes place as stated in the rollback algorithm. The termination of cascading effect is flow dependent on the correctness of transactions.
- Message sent and control type is delegate, the list of LSN delegated is the responsibility of delegatee transaction. The PrevLSN of message point log will point to the undelegated PrevLSN of the delegator.

Thus it is seen that algorithm presented here for rollback may require co-operation of recovery managers at more than one site for proper and flawless rollback of a given extended transaction.

3.2. Restart Procedure

The various types of failure which occur in distributed databases are transaction failures, communication failures, media failures and a set of site failures which include individual site, a group of sites or all the sites. A *transaction failure* occurs due to aborting a transaction, and recovery is done by partial or total rollback of transaction and is discussed in the previous subsection. A *communication failure* occurs due to the failure of communication links such that two sites can not communicate with each other. In our model we have assumed that network is FIFO and reliable, and any such failures are handled by computer networks. A *media failure* occurs due to the damage of nonvolatile storage, which homes database and the log. One common remedy of such failures is to have media mirrored on two different devices and at different locations to reduce probability of simultaneous failures. In this paper we assume that media survives a crash. In the event of site failure the contents of volatile storage are lost. The site failure can cause transactions on other site to be rolled back if they are recovery dependent on the crashed site.

The first action after a crash is to bring system back to a quasi-consistent state. Here quasi consistency means that active transactions can be rolled back to their latest save points to make them ready for re-execution. ARIES does this in three passes namely *analysis*, *redo* and *undo*. Since the messages are stored in stable storage before delivering, the message table survives the crash. Further after a message is received the communication manager sends the ack to communication manager at sending site only after it has stored the message in the incoming message table. Thus both Send and Receive Message Tables survive the crash. In the event of crash, recovery procedure initializes both dirty page table and active transaction table and puts back entries in them by repeating the history from *Begin_Chkpt* of the last completed checkpoint.

3.3. Analysis Pass

```
Restart_Analysis ( MasterRec );
  MasterRec is the LSN of Begin_Chkpt of last complete checkpoint */
  Initialize TrTable and DirtyPages to empty;
  LogRec:= Next_log; /* read log record following the Begin_Chkpt */
  WHILE NOT(End_of_log) Do;
  If transaction related record and LogRec.ResponsibleTrID 'NOT' in
  TrTable then {
    Insert relevant information from LogRec to TrTable;
    /* TrID, U, LastLSN, UndoNxtLSN */
    Initialize LastSavePoint ; /* this would be zero */
  }
  SELECT (logRec.Type)
  WHEN ('Update' or Comp) Do { ARIES actions };
  WHEN ('BgnChkpt') ; /* ARIES action, ignore */
  WHEN ('EndChkpt') Do {
    For each entry in LogRec.TrTable Do {
      If TrID NOT in TrTable then insert corresponding entry in TrTable;
      else if LastSavePoint in TrTable is less than corresponding
      LastSavePoint in LogRec.TrTable then update LastSavePoint;
    }
    For each entry in LogRec.DirtyPages Do { ARIES actions }
  }
  WHEN ('Msg') Do {
    TrTable[TrID].LastSavePoint:= LogRec.LSN;
  }
  WHEN ('Prepare' or 'Rollback') Do { ARIES actions};
  WHEN ('End') Do { ARIES actions};
  WHEN ('OsFileReturn') Do { ARIES actions};
  END SELECT;
  LogRec:=Next_log();
  END WHILE;
  Return;
```

Figure 6. The Analysis Pass

In proposed scheme analysis pass remains identical with ARIES except for the fact that the messages to be handled in the distributed extended long-lived transaction environment. The messages constitute possible save points. The analysis pass is used to track these for individual transactions. The adapted RESTART-ANALYSIS algorithm is

shown in the Figure 6. In the extended transaction environment the two special cases arise. They are (i) log record type message and (ii) log record type endcheckpoint. The processing of these cases provides latest save point for all active transactions. The active transactions with $LastSavePoint = 0$ will lose all the work as they have to be undone completely. Thus after the analysis pass all the active extended transactions get the most recent LastSave-Point which act as the bound for the undo pass later.

3.4. Redo Pass

The Redo pass is identical to the ARIES and messages have no role as redo pass takes database to the state just before the system crashed.

3.5. Undo Pass

```
Restart Undo(TrTable);
For all TrID in TrTable Do
  If (TrTable[TrID].Status = 'U') {
    SaveLSN:= TrTable[TrID].LastSavePoint;
    Rollback (SaveLSN, TrID);
    If ( SaveLSN := 0 ){
      LogWrite('End',..., 'Rollback')/*Resubmit the transaction*/
      Delete entry from TrTable corresponding to TrID;
    }
    else {
      Restore cursor position for application and lock data.
      Transaction ready for continuing execution.
    }
  }
End. /* end for all */
```

Figure 7. The Undo Pass at a Site

The Undo pass rolls back the active transactions at the time of crash in reverse chronological order. In the extended long-lived transaction model we are interested to save as much work as possible. Therefore we have incorporated the notion of save points. In the modified Undo Pass we rollback the transaction to latest save-point, get the cursor position of transaction application and start re-executing it. Figure 7 presents the Undo Pass algorithm which makes use of Rollback Algorithm.

As seen in the above algorithm if LastSavePoint is the beginning of the transaction then in undo pass the obituary of transaction is written else it is reexecuted from the save point onwards.

4. Case Studies

In this section case studies for different extended transaction models [5, 3, 4] have been discussed. These include nested, joint and split transactions.

4.1. Nested Transactions

In the nested transaction model parent child relationships exists among the transactions. The commitment of a child means delegation of responsibilities from the child to the parent. When the parent spawns a child it permits the child to access its view set i.e. the objects held and operations thereof. The parent is commit dependent on the child and the child is weak abort dependent on its parent. In an extended transaction scenario if a child aborts the parent has two options namely, (i) it can respawn the child with similar initial conditions, (ii) as it is unable to complete all operations due to death of the child abort. assigned operations it should abort.

4.2. Join Transaction

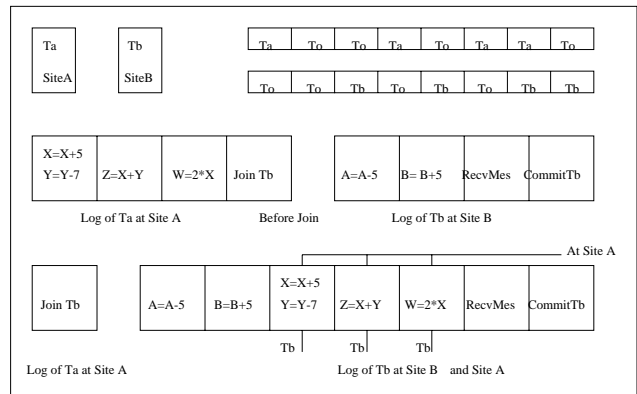


Figure 8. Join Transaction Model

Figure 8 refers to join transaction case. Transaction Ta joins transaction Tb. Join is a terminating event and thus responsibility of operations of Ta lies with Tb. If joint transaction Tb commits effects of Ta are made permanent otherwise they are discarded with that of Tb. In the Figure 8, To represents other transactions at the site. Ta Joins Tb through a message which is received by Tb as reflected in the log of Tb. It is to be noted that no log transfer is taking place but Tb has control over log of Ta for the operations it is responsible for. The control is managed through two communications managers and other DBMS components at the site A. Every Join is associated with a message and hence, the save point for joining and Joint transaction. The joining transaction can terminate once it is informed about successful delegation.

There are different variations of Join transaction namely Chain transaction, Report transaction and Co transaction. In chain transaction join remains a terminating condition. If we have a chain of N transactions with T_1 joining T_2 and

so on T_{N-1} joining T_N , the total number of message logs would be $2 * (N - 1)$. In the Report transaction model join ceases to be a terminating condition. Thus as against shown in Figure 8 the transaction T_a continues its operations and reports to transaction T_b and thus increasing responsibility of T_b . For every report one message is sent and every report is the save point. Thus, the maximum loss would be of one hours work for either of transaction if hourly report is send from one transaction to other. This also explains the utility of message save points. In the Co transaction model there is two way report meaning thereby that operations responsibility is transferred from one transaction to other. So in this case both the transactions would send and receive messages.

4.3. Split Transactions

In the split transaction model a transaction splits itself in two parts. One itself and the other is an independent subsidiary. In accordance with the splitting policy it can share its operations on the objects by delegating its subset to subsidiary. Subsidiary is abort dependent on its creator. In the Split transaction case, every split is associated with a message and thus a save point each for original transaction and its subsidiary.

5. Conclusion

In this paper we have presented a recovery mechanism for extended long-lived transactions in a distributed environment. We have taken ARIES framework as the basis and extensively modified its data structures and algorithms to produce a uniform recovery model suitable for above said environment of distributed extended long-lived transactions.

We have modeled distributed extended transactions through transaction primitives and messages. In the life of a transaction, a message event is the point of extensibility, the place of responsibility transfer and the position of control transfer in normal processing as well as in rollback and recovery. As the message event is an important location in normal processing and recovery of an extended transaction, it is chosen as the most suitable save point.

We have illustrated functionality of the system in normal circumstances. The algorithms for crash recovery and partial and total rollback in a distributed extended transaction environment have been presented and discussed in detail. In particular, we have demonstrated a uniform recovery mechanism for different kind of extended transactions in a distributed environment.

References

- [1] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison Wesley, 1987.
- [2] A. Biliris, S. Dar, N. Gehani, H. Jagadish, and K. Ramamritham. ASSET: A system for supporting extended transactions. In *Proc. of the 1994 ACM SIGMOD International Symposium on Management of Data*, pages 44–54, May 1994.
- [3] P. K. Chrysanthos and K. Ramamritham. Synthesis of extended transaction models using acta. *ACM Transactions on Database Systems*, 19(3):450–491, September 1994.
- [4] C. Pu, G. Kaiser, and N. Hutchinson. Split-transactions for open-ended activities. In *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 26–37. VLDB Endowment, 1988.
- [5] A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, 1992.
- [6] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.
- [7] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15:287–317, 1983.
- [8] J. Lin and M. Dunham. A survey of distributed database checkpointing. *Distributed and Parallel Databases*, 5:289–319, 1997.
- [9] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Transactions on Database Systems*, 17(1):94–162, March 1992.
- [10] C. Mohan and I. Narang. ARIES/CSA: a method for database recovery in client-server architectures. In *Proc. of the 1994 ACM SIGMOD International Symposium on Management of Data*, pages 55–66. ACM, May 1994.
- [11] E. Panagos and A. Biliris. Synchronization and recovery in a client-server storage system. *The VLDB Journal*, 6:209–223, 1997.
- [12] K. H. Prasad, T. K. Nayak, and R. K. Ghosh. DiET: A distributed extended transaction processing framework. In *Proc. of the 1996 International Conference on High Performance Computing*, pages 114–119. IEEE Computer Science Press, 1996.
- [13] R. Rastogi, P. Bohannon, J. Parker, A. Silberschatz, S. Shahadri, and S. Sudarshan. Distributed multi-level recovery in main-memory databases. *Distributed and Parallel Databases*, 6:41–71, 1998.
- [14] K. Rothermel and C. Mohan. ARIES/NT: A recovery method based on write-ahead logging for nested transactions. In *Proceedings of the 15th International Conference on Very Large Data Bases*, pages 337–346. VLDB Endowment, 1989.