ARIES/NT Modification for Advanced Transactions Support*

Henrietta Dombrowska
University of St.-Petersburg
E-mail: henr@orl2.usr.pu.ru

Abstract

A modification of ARIES/NT algorithm for nested transactions rollback and recovery is proposed. This modification allows to perform the *forward recovery* for advanced transaction models, such as ConTracts and some subclasses of Sagas. The implementation of the proposed algorithm at the storage system level is described.

1 Introduction

ARIES/NT [11] (ARIES for nested transactions) algorithm for transaction rollback and recovery is a simple and efficient recovery method. It uses write-ahead logging protocol and supports different modes of logging. Among the most important advantages of this algorithm are the ability to to attach different buffer management strategies, different lock modes and effective recovery: the system can start its activity after a crash when only the first step of recovery is completed.

ARIES/NT is an extension of ARIES recovery and concurrency control method introduced in [7] and inherits its major features. Though this method applies to a very general model of nested transactions and is flexible enough to be used for different modifications of transaction model, it is not sufficient to support some transaction models, namely those which require intermediate transaction recovery after the system crash.

The proposed modification of ARIES/NT algorithm allows to support a wide range of flexible transaction models and workflows. It was implemented using the buffer managing system of the storage system described in [9]. Some in-memory structures used by the buffer manager allow to simplify the basic ARIES scheme. The rest of the paper is organized as follows: in section 2 some important advanced transaction models are described. Section 3 gives a short description of the buffer manager of the storage system, its main functions and data structures, in section 4 the proposed modifications for ARIES/NT are considered. Conclusion shortly summarize the results.

*This work was partially supported by Russian Foundation for Basic Research under grant 95-01-00636, and UrbanSoft Ltd. under contract 35/95.

Proceedings of the International Workshop on Advances in Databases and Information Systems. Moscow, June 27–30, 1995.

2 Overview of advanced transaction models

The traditional transaction concept, which confirms the ACID transaction properties, ensures both execution and failure atomicity as it hides the effects of failures and concurrent processing. Such transactions are very effective in traditional applications which are characterized with short duration and access few data items.

However, in advanced non-traditional applications this traditional concept has limited applicability. Thus, in CAD/CAM, publication and software development environments transactions are usually very complex, access many data items and reside in the system for a long duration. Such *long-lived* transactions, introduced in [4], have high probability to be interrupted while their execution because of their long execution time. If such transaction will be rolled back due to failure atomicity requirement, much work will be lost. Besides, long lived transactions, due to isolation requirement, have to lock large amounts of data for long periods of time, leaving other transactions to wait for hours and days. Moreover, such transactions can easily cause deadlock.

Using traditional transactions in multidatabase systems and distributed databases also evokes some problems. These problems concern the global serialization and detecting of global conflicts which contradicts with the requirement of local autonomy.

To overcome some of the difficulties mentioned above several non-traditional transaction models were proposed, the most important of them will be shortly described below.

Two basic transaction models introduced in 80ths are conventional nested transactions and Sagas. In the nested transaction model introduced in [8], each transaction can initiate any number of nested transactions that are executed independently, but all the nested transactions should terminate (commit or abort) before the parent transaction. If the child transaction committs, its parent transaction can restart it, or ignore, or start the contingency subtransaction, or deside to abort. This model implies partial refusing from atomicity and durability.

In the Sagas model [3] each transaction is divided into relatively independent steps, for each of these steps the compensating transaction is defined. A system must guarantee that either the whole Saga will be executed or all its steps already performed will be compensated, so the committed steps may be rolled back. The locked data items are released at the end of each step.

Many other advanced transaction models are either concretizations of these two classes or combine some features of both of them. Thus, the DOM model [6] support different types and concretizations of nested transactions, among them decoupled and deferred transactions.

The ConTract model [10] requires *forward-recoverability*. This means that after a failure the ConTract must be continued from where it was interrupted. The Sagas extensions presented in the

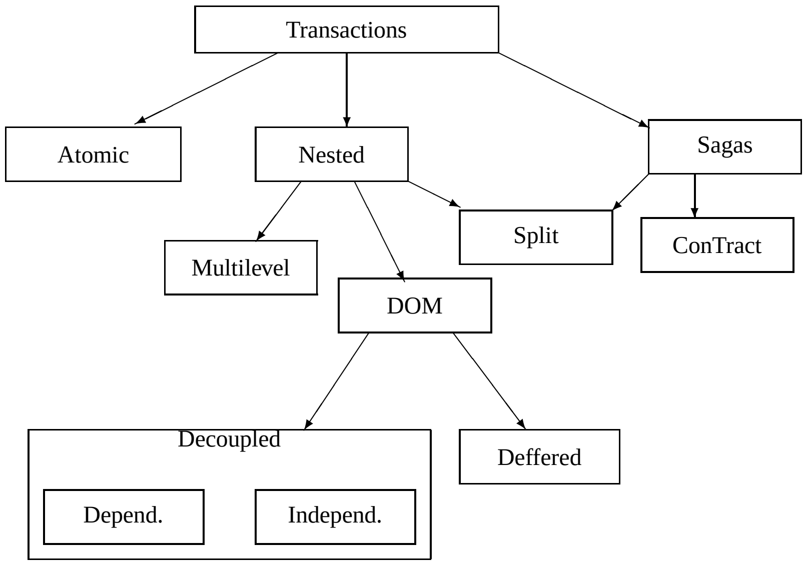


Figure 1: Advanced Transaction Classes

ACTA framework [1] allow to have nested Sagas and Sagas that can't be compensated. The *politransactions* [12] were developed to support interdependent data in multidatabase environment and allows to detect an existence of data dependencies during the transaction execution.

The hierarchy of advanced transaction classes is represented on figure 1.

3 Buffer management

The storage system was designed to support the manipulation of hierarchical complex data objects. It provides for representation, storing, indexing and searching of low-level representation of the SYNTHESIS [5] objects.

One of the most essential components of storage system kernel is the buffer manager, that supports reuse of blocks that are already read into memory, latching of the buffers and locking of the logical records, minimizing of the real input/output. It provides fine-granularity locking and supports the log, based on ARIES/NT algorithm for transaction rollback and recovery.

The functions of the buffer manager of the storage system are similar to those of many other systems of this type. It provides in-memory cache for disk storage. For the reasons of portability the storage system do not use hardware support for virtual memory. Explicit function calls to allocate, free, store pages to disk etc. are provided instead.

Each memory buffer is described internally by it's *header*, which includes the information used to support the reuse of blocks already read from disk, thereby minimizing the actual number of I/O operations and to provide the correctness of transaction access to the buffer. The buffer state reflects the buffer status and depends on the last operation performed with this buffer.

In the SYNTHESIS architecture a full-function concurrency control is independent of the storage system. The interoperable environment of heterogeneous information resources provides the logical transaction processing, including global distributed transactions which are subdivided into sets of local transactions over different resources. For this reason, the storage system provides only low-level transaction support ensuring physical data integrity, but do nothing about serialization, deadlock protection etc.

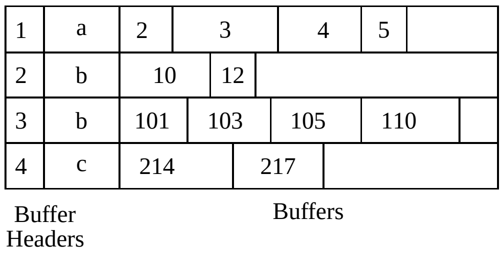
On the other hand, access methods provide logical operations on data items, such as *update, insert, delete*, so in the buffer system we don't consider garbage collection or growing records, but deal only with reading and writing of physical pages. For example, suppose that while performing *update* it will be discovered that the free space on the page is insufficient for the new value and the record should be transferred to some other page. The actions of the buffer manager will be the following: it will write to the stable storage the "old" page where the garbage collection will be already performed by the access method, obtain the new page, place there the updated record and write it to the stable storage. thus it will simply perform two updates.

As the buffer system provides fine-granularity locking, one buffer can be shared by several transactions. That's why for correct usage of the buffers the *Locks Table* is supported. For each record locked by transaction it contains the transaction code, the record number and the buffer identifier. Note that the previous discussion about logical and page-oriented operations can be completely applied in case of fine-grained locks.

If one of the records in a buffer should be updated by transaction that has locked it, this transaction latches the buffer, placing it's identifier to the buffer header. For the latched buffer this transaction is called it's *owner*. While the buffer is latched, only it's owner can access it (see figure 2). Note, that if the buffer is in the *free* state, it is considered to have no owner, and any transaction, which previously locked a records in this buffer can latch it.

In case of nested transactions each transaction can access all the records of it's direct or transitive parent transactions which are not currently locked, and is committed to it's parent while the real write-to-disk operation is executed after the up-level transaction commit. Such scheme allows to execute the transaction rollback at any level.

To provide the correct access to the buffers for nested transactions the Active Transaction Tree (ATT) is kept in memory [2]. Each node of this tree contains the transaction code and the pointers to the direct ancestor, to the elder brother and to the first child. It also contains the number of the last log record for each transaction, which is used for rollback during normal processing. Additional buffer system functions provide some operations on ATT, that allows to change its structure. These operations are used to support



Trans Id	Rec	BufId
a	2	1
b	103	3
a	101	3
c	214	4
c	5	1
b	12	2
b	3	1

Locks Table

Figure 2: Locking and Latching

other flexible transaction models, namely, decoupled and deffered transactions, split transactions, and some other.

4 Recovery method

.

ARIES is a family of algorithms for transaction rollback and recovery. It uses the widely accepted write ahead logging (WAL) protocol. This protocol asserts that the log records which represents the changes of the data items should be already on stable storage before the updated page will replace the previous version of that data. The recovery methods based on WAL protocol are used in such systems as IMS/VS, DB2, Starbust, NonStop SQL and some others.

The alternative methods are based on shadow page technique. To update a physical page its new copy is formed in the main memory, which is later transferred to disk as a *current version* of a page, while the previous copy is referred to as *shadow version*. This method has such well-known disadvantages as extra nonvolatile storage space overhead for shadow data copies, distributing the physical clustering of data and need for extra I/O operations for page map blocks modification.

Existing WAL-based methods differs in buffer management, ways they take normal checkpoints and usage of compensating log records. Most of them require specific buffer managing policy, locking granularity and so on. For example, DB2, NonStopSQL and some other systems adopted the steal and no-force policies. One of the advantages of ARIES is that it allows to attach different strategies of buffer management and supports a wide range of lock types. To achieve more efficiency, ARIES supports page-oriented *redo* and for high concurrency it supports logical *undo*.

In contrast to other WAL-based methods ARIES does not write log records to indicate the time when the data item was fetched to volatile storage. While normal processing it forms *undo* and *redo* records for each update operation and writes them into the log. All log records written by the same transaction are linked via a *backward chain*. The *LSN* – Last Sequent Number – identifies the record in the log and is used as a pointer in this chain. The checkpoints are taken during the normal processing as in other WAL-based methods.

In ARIES/NT method nested transactions should be supported. When a child transaction commits, the *child-commit* record is added into a backward chain of the parent transaction thus linking all its subtransactions chains into the backward chain tree.

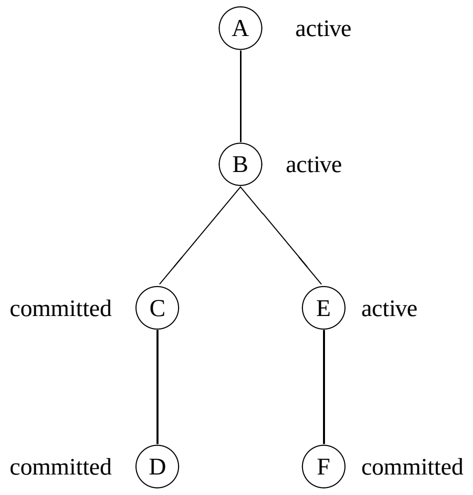


Figure 3: Nested Transaction Example

The piece of ARIES/NT log which corresponds to the execution of transaction A represented on figure 3 is shown on figure 4. We do not cover here such aspects of ARIES and ARIES/NT as adding *compensation log records* while transaction rollback, filling the log during recovery and some others.

After the system failure ARIES scans the log to extract the information about the pages that were updated in memory and not yet updated in the database; and about the transactions that were in progress at the time of a crash. Then ARIES repeats the history updating those records that were logged but not transferred to nonvolatile storage. During the last phase of recovery ARIES undoes the affect of all in-progress transactions. All the child transactions of in-progress top-level transaction are rolled back, no matter whether they were committed or not. For committed top-level transactions the lost updates of their subtransactions are repeated.

To support logging and recovery in the ordinary ARIES/NT algorithm several in-memory tables should be provided. The most important of them are *Transaction Table* and *Dirty Pages Table*. Both of them are used during normal processing and during restart recovery. In the latter case they are created using the information from the log records on the analysis pass of the log.

In considered buffer managing system the ARIES/NT algorithm was modified in two main aspects. The first concerns the support of in-memory structures. In the considered storage system all the information about dirty pages can be extracted from the buffer headers. Besides, the ATT nodes contain the codes, types and *Last LSN's* of in-progress transactions, so the system needn't support any additional in-memory structures.

The second aspect arises from the need to support a wide range of transaction models. Note that in ARIES/NT method the results of committed subtransactions are undone if their parent top-level

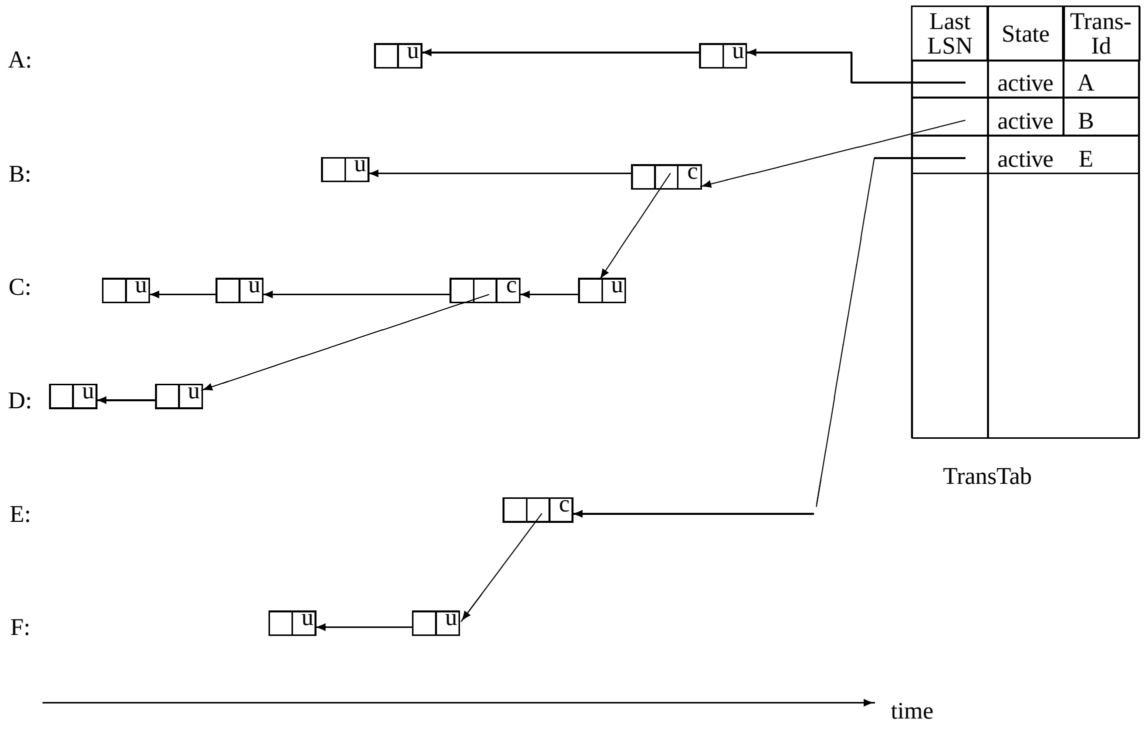


Figure 4: ARIES/NT Log Example

transaction hasn't been committed. Meanwhile, a large number of advanced transaction models require an ability to continue top-level transaction processing after a crash, that is require *forward recoverability*. In the proposed ARIES/NT modification new log record type *newtrans* is introduced which indicates the start of new transaction. The piece of the modified log is shown on figure 5.

Each record of *newtrans* type contain the subtransaction code, the parent transaction code and the subtransaction type. Due to this during the ananlysis pass the the ATT can be restored during the system recovery just in the same state as it was at the moment of a crash. This ability provides the way to support such flexible transaction models as ConTracts and nested sagas and a wide range of other advanced models supported by ACTA framework.

For the system with adavanced transaction models it's very difficult to define a performance measure and built appropriate model. On one hand, long-living activities can last for many hours. On the other hand, the execution time depends highly on application type. So, the only thing we can do is to compare the suggested recovery method with the basic ARIES/NT algorithm. Two things may be stated then: the first is that the proposed modification is correct relatively to the basic algorithm, and the next is that it is almost as efficient as ARIES/NT, because the only additional overhead is one more log record per transaction, which is normally less than few percents of the total number of records written to the log.

5 Conclusion

In this paper the ways of various types of transaction support by the buffer manager of the storage system where considered. The means to support fine granularity locking and the ARIES algorithm for transaction rollback and recovery were described. The main mechanism for nested transaction execution in the buffer manager is the active transaction tree. After inclusion of some additional information into it's nodes, the ATT can be used to support the whole range of flexible transaction models, thereby allowing to implement to all of them the ARIES algorithm.

The future research in this area will be concentrated on measuring performance of proposed recovery method and on comparing it with the techniques proposed in [13].

References

[1] P. Chrysanthis and K. Ramamritham. ACTA: A framework for specifying and reasoning about transaction structure and behavior. In *Proceedings of ACM SIGMOD Conference*, 1990.

[2] H. Dombrowska. The low-level support and logging for flexible transactions. In *Proc. of the Intnl. Workshop on Advances in Databases and Information Systems - ADBIS'94*, pages 49–53, Moscow, May 23–26 1994.

[3] H. Garcia-Molina and K. Salem. SAGAS. In *Proceedings of ACM SIGMOD Conference*, pages 249–259, San Francisco, California, May 1987.

[4] J. Gray. The Transaction Concept: Virtues and Limitations. In *Proceedings of the 7th International Conference on Very Large Data Bases*, pages 144–154, Cannes, France, September 1981.

[5] L. Kalinichenko. The interoperable environment of heterogeneous information resources: a generalization perspective. In *Proc. of the First International Workshop on the Interoperability in Multidatabase systems*, pages 196–199, Kyoto, April 1991.

[6] F. Manola and A. Buchmann. Functional/relational object-oriented model for distributed object management. Technical Report TM-0331-11-90-165, GTE Laboratories Inc., December 1990.

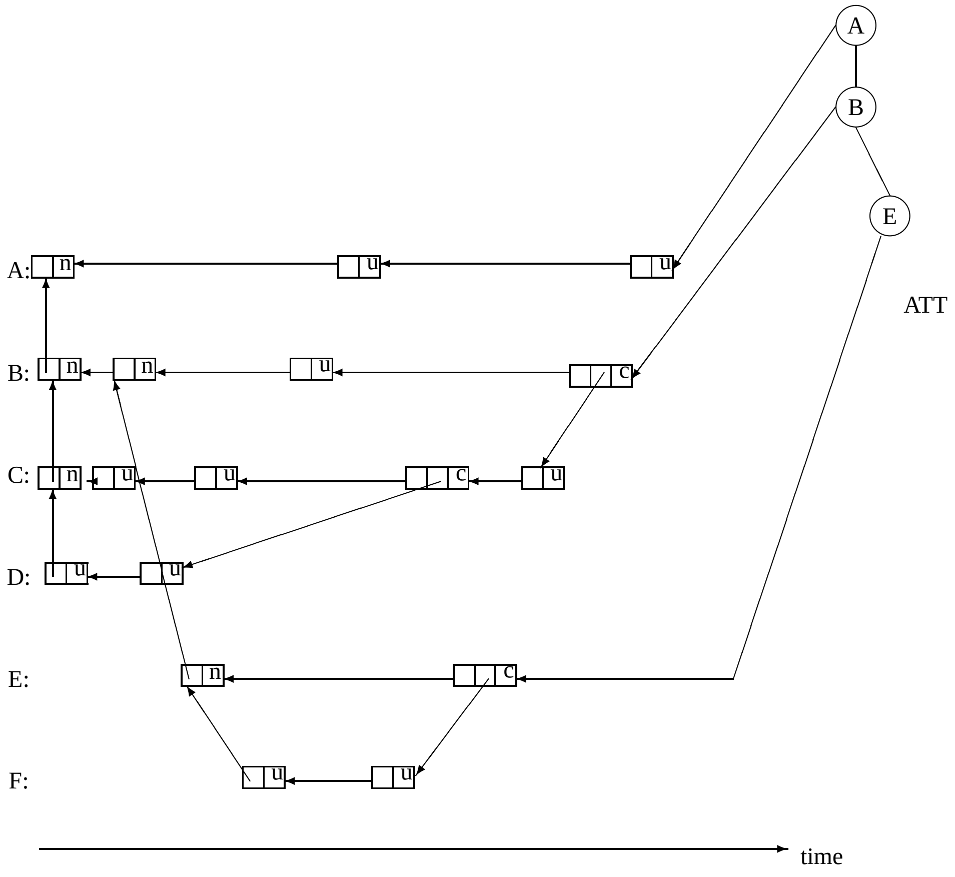


Figure 5: Modified Log Example

[7] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwartz. ARIES: A transaction recovery method supporting fine-granularity locking with partial rollbacks using write-ahead logging. *ACM Trans. on Database Systems*, 17(1):94–162, 1992.

[8] J. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing.* PhD thesis, MIT Press, Cambridge, MA, 1985.

[9] B. Novikov. The storage system support for complex objects. *USiM*, (7):46–52, July 1991.

[10] A. Reuter. Contracts: A Means for Extending Control Beyond Transaction Boundaries. In *Presentation at Third International Workshop on High Performance Systems*, September 1989.

[11] K. Rothermel and C. Mohan. ARIES/NT: A recovery method based on write-ahead logging for nested transactions. In *Proc. 15 conf. VLDB*, pages 337–346, 1989.

[12] M. Rusinkiewicz and A. Sheth. Polytransactions for managing interdependent data. *IEEE Data Engineering Bulletin*, 14(1), March 1991.

[13] E. Soisalon-Soininen and T. Ylonen. Partial strictness in two-phase locking. In *Proc. of the 5th Intnl. Conf on Database Theory - ICDT'95*, pages 139–147, Prague, January 11–13 1995.