The Low-Level Support and Logging for Flexible Transactions

Henrietta Dombrowska
Inst. for Math. and Mech. of St.-Petersburg university
E-mail: ban@niimm.spb.su

Abstract

In this paper the ways of various types of transaction support in the buffer system of the storage system are considered. Support of active transaction tree as the mechanism for nested transaction execution is described. It is shown that inclusion of some additional information into ATT nodes makes possible to support the wide range of flexible transaction concepts. The implementing of ARIES method for both flat and nested transaction rollback and recovery is described.

1 Introduction

The transaction processing is one of the most important components of distributed database systems. Recently the large number of flexible transaction models that do not confirm to ACID properties (atomicity, consistency, isolation and durability). The examples are split transactions, multi-level transactions, long-lived transactions, sagas, polytransactions and some others [1, 6]. The flexible transactions have complex internal structure reflecting the semantics of the application in the transaction design.

The basic (and the most simple) model of such sort of transactions is conventional nested transaction model. In this model 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. That causes the partial refusing from independence and durability, as the parent transaction termination depends on those of nested ones.

In this paper the mechanism for nested transaction support in the buffer management sub-system is described. The presented work was performed as a part of the storage system for the SYNTHESIS project, the basic research of which is concentrated around formally interoperable heterogeneous information resource environment. The main concern of the project are methods and tools for information resources representation and management.

The storage system [7] 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 objects.

One of the most essential components of storage system kernel is the buffer manager, that provides disk caching, latching of the buffers and locking of the logical records. It supports fine-granularity locking and supports the log, that is based on ARIES algorithm [2] for transaction rollback and recovery both for flat and nested transactions.

The rest of the paper is organized as follows: in section 2 the short description of the SYNTHESIS interoperability perspectives is presented. In section 3 the most important features of the buffer manager are outlined, and the buffer header and in-memory tables structure are described, in section 4 the ways of flat and nested transaction support on the buffer system level are considered, in section 5 the tree of active transactions and its usage for nested transactions support is described. Section 6 is devoted to other flexible transaction models implementation. In section 7 the correspondence between standard ARIES tables, the ATT and other in-memory tables of the buffer manager is stated. Conclusion shortly summarize the results.

2 The SYNTHESIS language

This storage system is developed as a part of the SYNTHESIS research project [4, 5], in which a prototype interoperable environment of heterogeneous information resources is being created. In SYNTHESIS interoperability means an ability to homogenize the heterogeneous information resource representation, an ability to organize the proper reuse of existing resources and an ability of dynamic design of an interoperable system capable of solving a given problem. For all these purposes the SYNTHESIS language was designed as a multipurpose language for equivalent homogeneous description of various heterogeneous information resources, information system conceptual design and application problem specification.

In implemented applications the problem of homoge-

Advances in Databases and Information Systems, ADBIS'94. Moscow, May 1994.

neous adequate specification of the the transactions confirming different models arises. The SYNTHESIS language provides general representation of different types of flexible transactions as the class hierarchy. It's support on the buffer level of the storage system is described in this paper.

3 Buffer Manager

The functions of 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 is described in [3]. The information maintained in the header is used to support the reuse of blocks already read from disk, thereby minimizing the actual number of I/O operations.

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 *Record Table* is supported. For each record locked by transaction it contains the transaction code, the record number and the buffer identifier. A separate buffer pool is reserved for log records. The *undo* and *redo* records are formed in this buffers after each record update, thier formats are defined by the access method. The log is forced to disk at end of transaction.

As it is shown in section 4, such buffer structure in couple with some additional in-memory tables allows to support both flat and nested transactions and the ARIES algorithm. Different buffer replacement strategies can be attached.

4 Multi-level and nested transactions

In the SYNTHESIS language the basic transaction class represents the flat single-level transactions. For representation of more complex activities various advanced transaction models, such as nested transactions, split transactions, and others, may be specified. All these transaction types are defined as hierarchy of subclasses of transaction class and reflect the nesting on the conceptual level of the application.

Besides, transactions as atomic operations can have different meaning on different levels of abstraction. In paper [10] the mechanism for multilevel transaction support and management of resulting nesting of transactions is described. An example of multilevel transaction for transferring a certain amount of money from one bank account to another is shown in figure 1.

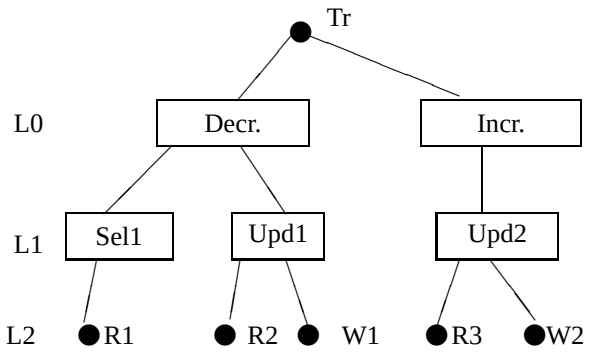

Figure 1: Multilevel Transaction

On logical level (L0) such transaction include two operations: test-and-decrement on source account and increment on target account (*Incr* and *Decr*). This logical operations are implemented on the tuple level (L1) by means of tuple operations *Select* and *Update* and finally are transformed into page access operations on the page level (L2).

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.

Thereby in field of transaction processing support the storage system should provide:

- the correct transaction commit or abort;
- the correct usage of the buffers.

The correctness of transaction commit means that simultaneously with transaction commit all updates are fixed in the stable storage. If it is impossible for some of transaction records due to the fact that the buffer that contain it is latched by some other transaction, the updates are fixed in the log according to the WAL protocol and the buffer is marked to be written to disk after releasing. At transaction abort the backup of transaction is provided.

Supporting of the log and existense of both locking and latching mechanisms provides the correct access of transactions to the buffers. At transaction abort the rollback of transaction is provided.

In case of nested transactions each transaction can access all the records of it's direct or transitive parent transactions which are not currenyly latched, and is committed

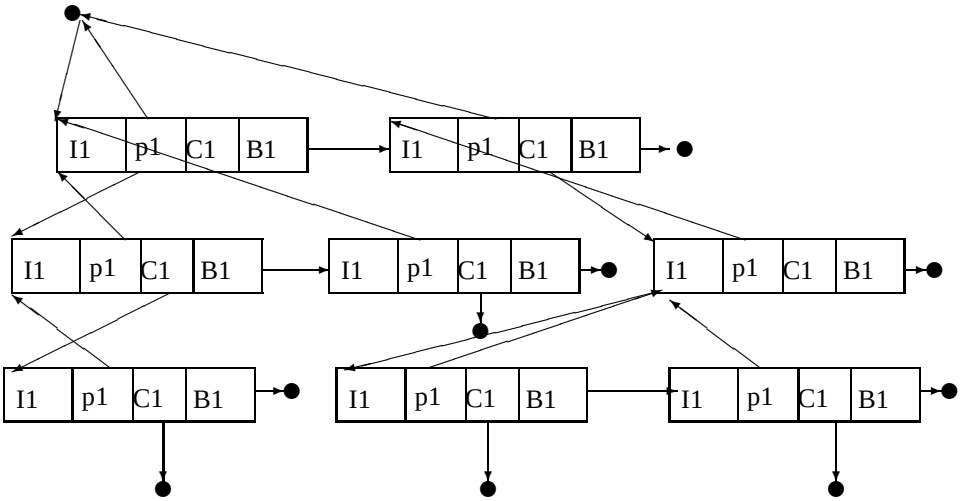

Figure 2: Active Transactions Tree

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.

5 The Tree of Active Transactions

For nested transaction support the tree of active transactions is kept in memory. 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 userd for rollback during normal processing. The transaction tree example is shown on figure 2.

Such structure allows efficient implementation of all operations necessary for transaction support, as described in [3].

Here the actions performed by the buffer manager for some of these operations will be described.

While obtaining the buffer for reading or forming a new page contents the new entry is added to the Record Table, which include transaction code, the record number and the buffer address. This means the logical locking of this record with the current transaction. If one of the records in a buffer is to be updated by transaction that has locked it, this transaction latches the buffer, placing it's identifier into 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, so at each buffer access the current transaction identifier is compared with it's owner identifier. If it isn't, the correctness of access is checked using the Record Table, asuming that the correctness of accessing the record inside the buffer is left to the access method.

In case of nested transactions the place of current transaction in the ATT should be found. If the buffer is latched, the access to it is block for all the transactions exept it's owner. In other case the access to the buffer is correct if either the current transaction or it's direct or transient parent in the ATT has the locked records in this buffer.After the buffer is updated, and thereby the record is added to the log, the last log record number is replaced in the ATT.

At transaction commit the termination of all it's created transactions (i. e. absence of descendants) is checked, the Record Table is modified so that all the records locked by this transaction are set to be locked by it's parent and the corresponding node is deleted from the ATT. The real write-to-disk operation is performed only for top-level transaction, if it is not required by the replacement strategy or forced by some other transaction commit.

At top-level transaction abort the rollback of all created transactions starting from the lowest level is performed. When one of the nested transaction is aborted, it does not affect the parent transaction, which makes a decision about it's restart or rollback.

6 Other Flexible Transaction Models

The SYNTHESIS language support some other flexible transaction classes based on the conventional nested transaction model. All of them can be implemented using some modifications of ATT.

The model for long-living activities allows the created transaction to be performed not as a child of it's creator, but as a child of another transaction, that is called the proper parent. When the child is running, it's proper parent is suspended.

To support this model the node in ATT that corresponds to created transaction is moved to become a child of another parent, while all buffer operations are executed as in case of conventional nested transactions.

Two other possible extensions are *deferred* and *decoupled* transactions. The execution of deferred transaction is delayed until the end of the top transaction, and all of them are considered to be independent. If in their processing some new deferred subtransactions will be created, they will be delayed till all transactions of the previous cycle will be ended.

To support this model, some additional information should be included into each ATT node, that is the cycle number. This field is assumed to be NULL for conventional nested transactions. When a deferred subtransaction is created, it is placed in the ATT not as a child of it's parent, but as a top-level transaction, and the cycle number is set next to the creator's one. At any access to buffer of the deferred transaction the system checks whether all the previous cycle transactions are ended, examining the status of the ATT. The scheme of ATT with deferred transaction is presented on figure 3.

In case of *decoupled* transaction model the created transaction is represented by it's own tree and is considered to be concurrent to it's proper parent if it is *causally-independent.* If it is *causally-dependent,* it should be serialized after it's proper parent.

To implement this transaction model at start of decoupled transaction the corresponding node is inserted into the ATT not as a child of it's creator, but as a child of it's creator direct parent which becomes it's proper parent. The field that contains the cycle number in case of deferred transaction can be used to keep a flag which indicates if transaction is causally-dependent or independent. The scheme of ATT with decoupled transaction is presented on figure 4.

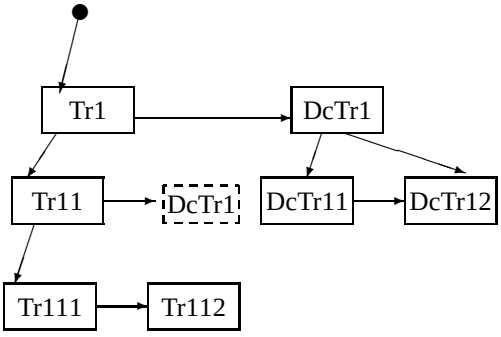

Figure 4: Decoupled Transaction

Another transaction model to be mentioned is split transaction model that allows splitting of processing transaction into several independent ones. This ability can be implemented by inserting additional nodes into ATT instead of the splitted transaction and changing transaction identifiers in it's buffers as it is defined by transaction.

Basing on all variety of nested transaction models and the split transaction model, the *polytransaction* model can be defined as a "transitive closure" of a transaction submitted to an independent data management system. The tree of such transaction is built according to interdatabase dependency and mutual consistency predicates of Data Dependency Descriptors [9]. There are several models of execution of the child transactions, that were described in the current section.

7 Implementation of ARIES

In standard ARIES/NT [8] (ARIES for nested transactions) implementations several in-memory tables should be provided for rollback and recovery support. 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 the considered storage system the ATT contains all the information about transactions and the information about dirty pages can be extracted from the buffer headers.

At start of nested transaction the corresponding record that indicates the parent transaction identifier and transaction type is added to the log. The log records of the special type indicate the transaction operations, i.e. decoupling and deffering (see section 6). The undo and redo records contain, exept the data itself, the information from the buffer header.

8 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.

References

[1] D. Agrawal and A. E. Abbadi. A Non-Restrictive Concurrency Control for Object Oriented Databases. In *Extending Data Base Technology. Proc. Int. Conf.,*

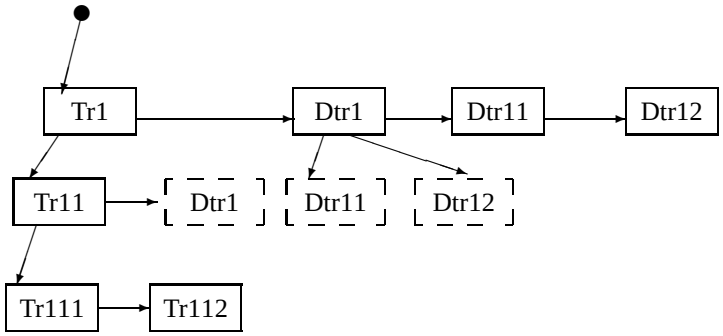

Figure 3: Deferred Transaction

EDBT-92, volume 580 of *Lect. Notes in Comp. Sci.*, pages 469–481, Vienna, Austria, March 1992.

[2] M. C., H. Don, L. Bruce, P. Hamid, and S. Peter. 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.

[3] H. Dombrowska. Nested transaction support by the storage management system in the SYNTHESIS project. In *Proc. of the workshop on advances in databases and information systems - ADBIS'93*, pages 81–85, Moscow, May 22–24 1993. (in Russian).

[4] 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.

[5] L. A. Kalinichenko. *SYNTHESIS: a language for descriptive design and programming interoperable information resource management*. Institute of Problems of Informatics of the USSR Academy of Sciences, September 1991. (in Russian).

[6] Y. Leu, A. Elmagarmid, and N. Boudriga. Specification and execution of transactions for advanced database applications. *Inf. Syst.*, 17(2):171–183, 1992.

[7] B. Novikov. Temporal database indexing. In *Proc. of the workshop on advances in databases and information systems - ADBIS'93*, pages 69–74, Moscow, May 22–24 1993. (in Russian).

[8] 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.

[9] M. Rusinkiewicz, A. Sheth, and G. Karabatis. Specifying interdatabase dependencies in a multidatabase environment. *Computer*, 24(12):46–53, 1991.

[10] G. Weikum. Principles and realization strategies of multilevel transaction management. *ACM Trans. Database Syst.*, 16(1):132–180, 1991.