ELSEVIER

The Journal of Systems and Software 55 (2000) 151–165

The Journal of Systems and Software

www.elsevier.com/locate/jss

An open and safe nested transaction model: concurrency and recovery

Sanjay Kumar Madria [a,*], S.N. Maheshwari [b], B. Chandra [c], Bharat Bhargava [a]

[a] Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA
[b] Department of Computer Science and Engineering, Indian Institute of Technology, Hauz Khas, New Delhi, India
[c] Department of Mathematics, Indian Institute of Technology, Hauz Khas, New Delhi, India

Received 1 August 1999; received in revised form 1 November 1999; accepted 8 December 1999

Abstract

In this paper, we present an open and safe nested transaction model. We discuss the concurrency control and recovery algorithms for our model. Our nested transaction model uses the notion of a recovery point subtransaction in the nested transaction tree. It incorporates a prewrite operation before each write operation to increase the potential concurrency. Our transaction model is termed "open and safe" as prewrites allow early reads (before writes are performed on disk) without cascading aborts. The systems restart and buffer management operations are also modeled as nested transactions to exploit possible concurrency during restart. The concurrency control algorithm proposed for database operations is also used to control concurrent recovery operations. We have given a snapshot of complete transaction processing, data structures involved and, building the restart state in case of crash recovery.

1. Introduction

1.1. Overview of nested transaction models and recovery algorithms

1.1.1. Close nested transaction model

In close nested transaction model (Moss, 1985), a subtransaction may contain operations to be performed concurrently, or operations that may be aborted independent of their invoking transaction. Such operations are considered as subtransactions of the original transaction. This parent–child relationship defines a nested transaction tree and transactions are termed as nested transactions (Moss, 1985). Failure of subtransactions may result in the invocation of alternate subtransactions that could replace the failed ones to accomplish the successful completion of the whole transaction. Each transaction has to acquire the respective lock before accessing a data object. A subtransaction's effect cannot be seen outside its parent's view (hence, called closed). A child transaction has access to the data locked by its parent. When a transaction writes a data object, a new version of the object is created. This version of the object is stored in volatile memory. When the subtransaction commits, the updated versions of the object are passed to its parent. If the transaction aborts, the new version of the object is discarded. Parent commits only after all its children are terminated. When the top-level transaction commits, the current version of each object is saved on stable storage.

In the closed nested transaction model, the availability is restricted as the scope of each subtransaction is restricted to its parent only. This forces a subtransaction to pass all its locks and versions of data objects updated to its parent on commit. The effect of a committed subtransaction is made permanent only when the top-level transaction commits. In many applications, it is unacceptable that the work of a longlived transaction is completely undone by using either of the above techniques in case the transaction eventually fails at finishing stage. The current strategy forces short-lived transactions to wait before they acquire locks until the top-level transactions commit and release their locks. Therefore, the model is not appropriate for the system that consists of long and short transactions.

* Corresponding author. Present address: Department of Computer Science, University of Missouri, Rolla, MO 65405, USA.

E-mail addresses: madrias@umr.edu, skm@cs.purdue.edu (S.K. Madria), snm@cse.iitd.ernet.in (S.N. Maheshwari), bchandra@maths.iitd.ernet.in (B. Chandra), bb@cs.purdue.edu (B. Bhargava).

0164-1212/00/$ - see front matter
PII: S0164-1212(00)00067-4

1.1.2. Open nested transaction model

To exploit layer specific semantics at each level of operation nesting, Weikum presented a multilevel transaction model (Weikum, 1991; Weikum et al., 1990). The model provides non-strict execution by taking into account the commutative properties of the semantics of operations at each level of data abstraction, which achieves a higher degree of concurrency. A subtransaction is allowed to release locks before the commit of higher level transactions. The leaf level locks are released early only if the semantics of the operations are known and the corresponding compensatory actions defined. When a high level transaction aborts, its effect is undone by executing an inverse action which compensates the completed transaction. Recovery from system crashes is provided by executing undo actions at the upper levels and redo actions at the leaf level. Each level is provided with a level specific recovery mechanism. This model has also been studied in the framework of object oriented databases in Muth et al. (1993) and Resende et al. (1994).

In many applications, the semantics of transactions may not be known and hence, it is difficult to provide non-strict executions. In real time situations, there are other classes of operations that cannot be compensated. These are the operations that have an irreversible external effect, such as handing over huge amounts of money at an automatic teller machine. Such operations have to be deferred until top-level commits, which restricts availability (i.e., increases response time).

1.1.3. Nested transaction recovery algorithms

The intentions-list and undo-logging recovery algorithms given in Fekete et al. (1993) handle recovery from transaction aborts in the nested transaction environment by exploiting the commutative properties of the operations. The intentions-list algorithm works by maintaining a list of operations for each transaction. When a transaction commits, its list is appended to its parent; when it aborts, the intentions-list is discarded. When the top level transaction commits, its intentions-list is transferred to the log. This scheme provides recovery from transaction aborts only and does not handle system crashes. To increase concurrency during undo logging recovery, scheme allows some non-strict executions. It allows a transaction to share the uncommitted updates made by other transactions by exploiting commutativity of operations. On execution of an operation, the data object records change their states and the new state is transferred to the log. When a transaction aborts, in contrast to intentions-list algorithm, all operations executed by its descendants on the object are undone from its current state and are also subsequently removed from the log. This algorithm does not take care of recovery from system crashes.

In both intentions-list and undo-logging algorithms, an incomplete transaction is allowed to make uncommitted updates available to those transactions that perform a commutative operation. However, this is restricted to transactions at the same level of abstraction. This limits availability. In both algorithms, all the work done by descendent transactions are discarded in case of aborts at higher levels. This may not be possible or desirable in many real time applications. In undo-logging algorithm, when a transaction aborts, in contrast to intentions-list algorithm, all operations executed by its descendants on the object are undone from its current state and are subsequently removed from the log. In both intentions-list and undo-logging algorithms, an incomplete transaction is allowed to make uncommitted updates visible to those transactions that perform a commutative operation. This is restricted to the transactions at the same level of abstraction.

The above two recovery models consider semantics of operations at leaf level only. System R (Gray et al., 1981) exploits layer specific semantics but restricted to two level of transaction nesting. In System R, to perform recovery, updates are undone by performing inverse tuple-level operations. For this purpose, System R records tuple updates on a log. To recover from a system crash, before applying any tuple level log record, the database must first be restored to some tuple-level consistent state. In other words, a low-level recover mechanism is necessary to make tuple actions appear atomic.

In Moss (1987), a crash recovery technique similar to shadow page has been suggested in nested transaction environment based on undo/redo log methods. In terms of logging, both undo/redo logs are used. Mohan et al. (1992, 1989) has also discussed "write ahead logging" based crash recovery algorithm using conventional nested transaction model. This undo/redo type of recovery model exploits semantics of nested transactions. The actions of a transaction undone during previous abort have not been undone again in case of one more failure. This is an advantage over Weikum's multilevel recovery algorithm by which requires undo actions to be undone again in case of one more failure.

1.2. Our contributions

In this paper, we introduce an open and safe nested transaction model in the environment of normal read and write operations to remove the deficiencies stated above and to further improve availability and provide efficient crash recovery. Our model supports inter- and intra-transaction concurrency. We assume that semantics of transactions at various levels of nesting are not known. There are two basic motivations behind our model. First, it is desirable that long-lived transactions should be able to release their locks before top-level