ELSEVIER

Information and Software Technology 42 (2000) 445–464

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.nl/locate/infsof

Multi-level transaction model for semantic concurrency control in linear hash structures

S.K. Madria[a,*], M.A. Tubaishat[b], B. Bhargava[a]

[a]Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA
[b]School of Computer Sciences, Universiti Sains Malaysia, 11800 Penang, Malaysia

Received 10 October 1998; received in revised form 18 August 1999; accepted 14 December 1999

Abstract

In this paper, we present a version of the linear hash structure algorithm to increase concurrency using multi-level transaction model. We exploit the semantics of the linear hash operations at each level of transaction nesting to allow more concurrency. We implement each linear hash operation by a sequence of operations at lower level of abstraction. Each linear hash operation at leaf-level is a combination of search and read/write operations. We consider locks at both vertex (page) and key level (tuple) to further increase concurrency. As undo-based recovery is not possible with multi-level transactions, we use compensation-based undo to achieve atomicity. We have implemented our model using object-oriented technology and multithreading paradigm. In our implementation, linear hash operations such as find, insert, delete, split, and merge are implemented as methods and correspond to multi-level transactions. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Linear hash structure; Concurrency; Multi-level transaction; Object-oriented; Three-tier client/server; Multithreading

1. Introduction

To exploit layer specific semantics at each level of operation nesting, Weikum presented a multi-level transaction model [1,17]. The model has received considerable attention for designing high performance transaction processing and concurrency control algorithms [14,15]. The multi-level transaction model takes into account the commutative properties of the semantics of operations at each level of data abstraction to achieve a higher degree of concurrency. If two operations at the same high level commute then their conflicting descendants at the same lower level will be allowed to execute since they will not introduce any inconsistencies. In this model, a subtransaction is allowed to release a lock on finishing before the commit of high level transactions. In case a subtransaction aborts, the aborted subtransaction's effect is undone by a compensatory transaction. Two actions at the lower level of abstraction conflict if there is a state in which the corresponding actions do not commute. Multi-level transaction models enhance advanced DBMS applications by providing better support for long-lived activities, capture more semantics about the operations and enhance parallelism.

Attention is being given to the designing of concurrency control algorithms which take advantage of the knowledge of particular data structures and the semantics of operations such as insert, delete, find, etc. to improve availability and expedite accesses. Data structures that have been studied with the above in mind are B-trees [8,16] and hashing techniques [2,3,5–7,12]. Hashing is one of many addressing techniques used to find a record based on its unique key value. When a record is to be updated, the system takes the value of the key and performs some type of calculations to derive a target address for the record. Among various hashing techniques, Linear Hashing [2,5] is interesting because of its simplicity and ability to handle large amount of data with increased concurrency, and speed in retrieving data.

Current database applications are increasingly using object-oriented techniques that necessitate the revaluation of the traditional concurrency control methods that are in use for more efficiency. When combining database functionality with object-oriented concepts, object-oriented databases become the ideal repository of the information that is shared by multiple users and multiple applications on different platforms. In object-oriented database, if the execution of methods is seen as a transaction then the

* Corresponding author. Fax: + 1-765-494-0739.
E-mail addresses: skm@cs.purdue.edu (S.K. Madria); malik@cs.usm.my (M.A. Tubaishat); bb@cs.purdue.edu (B. Bhargava).

execution of methods generates a hierarchy that corresponds to nested transactions.

Nested transaction [13] processing and concurrency control issues play a major role in providing higher concurrency and better handling of transaction's abort and hence have been an important area of research in database systems. In Ref. [4], B-tree algorithm in nested transaction environment has been presented to show its correctness but no implementation issues have been discussed. In Ref. [11], linear hashing using nested transactions has been studied only with the aim of formalizing and proving the correctness of the algorithm using the I/O automaton model [10]. While it is essential that object-oriented database systems must support multiple concurrent users, to our knowledge, no implementation of multi-level transactions accessing linear hash structure has been available in the literature.

In this paper, we exploit the semantics of multi-level transactions at each level of nesting in a linear hash structure environment. We allow various multi-level transactions to execute in parallel and make the effects of the subtransactions persistent before the commit of top-level transaction. However, we execute a compensating transaction [17] in case the parent transaction of a previously committed subtransaction aborts. Thus, we handle transaction aborts in our model. We consider the granularity of data mainly at two levels; one at the bucket level and the other at the key level. Thus, we manage the locking at two levels, which results in more concurrency. We have realized that in the case of aborts of restructuring operations split or merge, there is no need of executing any compensated transaction. The reason being these operations affect only the structure of the data, but do not change the key values stored therein. These types of operations are usually referred to as nested top-actions [12].

We have implemented a multi-level transaction version of the linear hash structure algorithm [2] using object-oriented concepts and multithreading paradigm. In our implementation, we model buckets (pages) as objects and linear hash operations find, insert, delete, split, and merge as methods. These methods correspond to multi-level transactions and are implemented as multithreads. Multithreading is a programming paradigm that allows the application processes to run concurrently. When the process performs multiple tasks at the same time, multithreads split themselves into separate threads that run concurrently and independently. Each thread performs one atomic action. The threads execute individually and are unaware of the other threads in a process. Thus, multithreading can be used very efficiently to implement the behaviors of multi-level transactions.

Finally, we have designed and implemented our system using layered system architecture in three-tier client/server environment, which allows more flexibility in terms of decomposing of application programs whose modules may be designed and implemented independently.

Rest of the paper is organized as follows. Section 2 outlines some preliminary discussion on multi-level transactions and on linear hash structure, concurrency control, and locking. Section 3 discusses the linear hash structure model in multi-level transaction environment. Section 4 discusses various scenarios of multi-level transactions in linear hash structure environment. In Section 5, we discuss the layered system architecture for the implementation of our model. Section 6 explains the object-oriented design of our model. Section 7 describes methods monitor between middlewave and clients. Section 8 discusses methods monitor between middlewave and database. Section 9 describes the multi-level transactions using multithreading in the form of a case study. We conclude in Section 10.

2. Preliminaries

2.1. Principle of multi-level transactions

A multi-level transaction is a balanced tree of actions in which all nodes at the same depth correspond to operations of the same level of abstraction. Edges in a transaction tree present the implementation of an operation by a sequence of operations at the same lower level. In multi-level transaction model, transaction management takes place on a fixed set of levels simultaneously. It is particularly suited to the design and description of transaction management in complex systems. In the layered system, transaction management is distributed among the levels. Multi-level transactions are a variant of open-nested transactions in which the subtransactions correspond to operations at different levels of layered system architecture.

The transaction aborts in this model can no longer be implemented by restoring the before-image (pre-transaction state) of the modified objects which may result in illegal effects such as unintentionally backing out updates of a possibly already committed transaction. In this model, committed subtransactions need to be compensated by means of appropriate "inverse" operations. The necessary compensating subtransactions are again subject to the semantic concurrency control protocol.

The locks of the actions in a subtransaction are released upon the completion of the subtransaction, rather than being passed to the subtransaction's parent as in the conventional model of "closed nested transaction" [13]. Only the transaction root holds semantic locks for its children (i.e. top-level actions) until the end of the transaction. This means that subtransactions commit expose their effects independently of the commit of their ancestors. However, because the parent of a committed subtransaction holds a higher-level semantic lock, the effects of the subtransaction are visible only to such actions that commute with the root of the subtransaction. This locking protocol ensures semantic serializability and is equivalent to a serial execution as viewed by the top-level actions of the executed transactions.

In principle, one may adopt a uniform approach to