Context-Specific Synchronization for Atomic Data Types *

M. H. Wong and D. Agrawal

Department of Computer Science
University of California, Santa Barbara, CA 93106

Abstract. Highly concurrent and reliable atomic data types are crucial for the design of distributed systems. Deferred update (DU) and update-in-place (UIP) are two common recovery strategies for implementing atomic data types. These two strategies place incomparable constraints on the conflict relations between concurrent operations resulting in incomparable synchronization protocols. Also, the conflict relations used are usually static in the sense that the algorithms do not use the context-specific information that may be available in the system. In this paper, a new synchronization mechanism that employs a hybrid recovery scheme by using both DU and UIP is proposed. Furthermore, the protocol is dynamic in the sense that the context-specific information is also used to determine conflict relations among concurrent operations on the atomic data types. Another extension is the use of ordered shared relationship between locks to execute conflicting operations concurrently. Thus, the execution of operations is never delayed in the proposed protocol, however, the commitment of the transactions invoking these operations may be delayed due to the restriction imposed by the ordered shared relationship between locks. We also demonstrate that the sets of histories accepted by a two phase locking protocol using either DU or UIP are proper subsets of the set of histories accepted by the proposed protocol.

1 Introduction

Atomic transactions are widely used for coping with concurrency and failures in database systems. Concurrency control and recovery are two of the main components for transaction management in databases. Concurrency control ensures the correct execution of a set of transactions, even when the operations of different transactions are interleaved. Recovery ensures the consistency of the database state, even when failures occur, or when transactions abort before completion. Traditionally, database systems are modeled as a collection of objects which can only be read or written by transactions [9, 3, 2]. More recently, a number of researchers have considered placing more structure on data objects and have shown how this structure can be used to permit more concurrency [7, 10, 11, 6, 13, 14, 8, 5, 4]. The notion of atomic transactions is also used to implement atomic data types in distributed systems. In particular, the system is modeled as a collection of objects with abstract data type

* This research is supported by the NSF under the grant numbers IRI-9004998 and IRI-9117094.
specifications and synchronization algorithms for concurrency control and recovery exploit the semantics of data types.

A theory for analyzing the interrelationship between concurrency control and recovery protocols is developed in [14]. The author uses two phase locking as an example of a concurrency control protocol and deferred update (DU) and update-in-place (UIP) strategies as examples of recovery schemes. He then shows that the two recovery schemes place incomparable constraints on concurrency control resulting in incomparable synchronization protocols. The protocols presented in [14] are static in the sense that the algorithms do not use the context-specific information that may be available in the system. Consider an example of a banking system with a locking protocol that uses update-in-place for recovery. In such a system, a successful withdraw operation conflicts with all concurrent deposit operations, since a withdraw operation does not in general commute with a deposit operation when they are executed with the update-in-place strategy. Consider the following execution of operations of three different transactions on an account with initial balance of $0:

| Transaction 1: | deposit($10) |
| Transaction 2: | deposit($10) |
| Transaction 3: | withdraw($8) |

If we use the static notion of conflicts, the withdraw operation conflicts with both deposit operations in the above execution. However, if the context of the operations is also taken into account, the withdraw operation may conflict only with the deposit operation of the first transaction, but not with the deposit operation of the second transaction. In other words, if it is sure that the first deposit operation is serialized before the withdraw operation, there is no constraint on the serialization order of the second deposit operation with respect to the withdraw operation. In this paper, we develop a protocol that employs a dynamic notion of conflict by using context-specific information for executing operations.

We use a hybrid recovery scheme based on both deferred update and update-in-place. Thus, in the above example the withdraw operation is executed with update-in-place strategy with respect to the first deposit operation while it is executed in deferred update strategy with respect to the second deposit operation. From [14], it can be verified that indeed, the withdraw operation conflicts (since it does not right commutes backward) with the first deposit operation while it does not conflict (since it forward commutes) with the second deposit operation.

Another extension that is used in this paper is that of using ordered shared relationship between locks with two phase locking [1]. Note that if we employ the standard shared and exclusive relationships between locks to capture the no-conflict and conflict relations between operations, in the above example the withdraw operation cannot be executed concurrently with the two deposit operations since it conflicts with at least one deposit operation. Ordered shared relationship between locks permits concurrent execution of conflicting operations as long as the serialization order of the transactions is consistent with respect to the ordered shared execution of conflicting operations.

The paper is organized as follows. In the next section, we present the model of the system. In Section 3, we present the specification and implementation of the proposed protocol. In section 4, we present the proof of correctness of the proposed