Towards a Comprehensive Concurrency Control Mechanism for Object-Oriented Databases

David H. Olsen
University of Akron

Sudha Ram
University of Arizona

Object-Oriented databases are becoming increasingly popular in business. Issues such as query optimization, analysis and design techniques, and concurrency control have been addressed as they pertain to the relational model but have not been addressed as they apply to the object-oriented model. This paper includes the framework development and description of a concurrency control mechanism named \(O^2C^2\) which is specifically designed for an object-oriented database. \(O^2C^2\) is a lock–based concurrency control mechanism that forms the basis of this research.

A description of database concurrency control and object-oriented database precepts are presented to provide a basis for a comprehensive framework for concurrency control in object-oriented databases. The theory is developed along four specified dimensions which are the hierarchical level dimension, the data type dimension, the composite or complex objects dimension and transaction type dimension. Additionally, a comprehensive list of rules is given that are crucial to an object-oriented database concurrency control mechanism. The rules are given to provide a basis not only for the \(O^2C^2\) mechanism, but for any object-oriented database concurrency control mechanism. The \(O^2C^2\) mechanism is then presented after which a discussion ensues about the possible transaction types in order to demonstrate the robustness of the mechanism.

Concurrency control (CC) is defined by Bernstein et al. (1987) as follows: “concurrency control is the activity of coordinating the actions of processes that operate in parallel, access shared data, and therefore potentially interfere with each other.” In other words, CC mechanisms interleave the operations of competing processes in such a way that consistency is maintained. It is also one of a few critical components of a database management system. A few basic approaches to concurrency control have been proposed and developed in the database area. Based on these approaches, several hundred of algorithms have been created (Barghouti & Kaiser, 1991; Bernstein, et al., 1987). However, very few of these address object-oriented databases.

The concurrency control mechanism \(O^2C^2\), described in this research contributes in two important ways. First, the concurrency control mechanism \(O^2C^2\) is presented along with how it relates to the developed framework. \(O^2C^2\) is lock–based and two–phased but is more complex than standard algorithms in order to deal with the complexities of object–oriented databases. Second, it adds to the body of theory that explains the objectives, problems, and tradeoffs in the area of concurrency control for object–oriented databases by comparing it to the mechanisms in ORION and \(O_2\).

The advantages of \(O^2C^2\) include a higher degree of concurrency than current implementations. It is superior to other approaches because more transactions are allowed to process at the same time due to fewer conflicting lock types and a finer granularity of lock types. In a majority of circumstances, the higher degree of concurrency yielded by \(O^2C^2\) results in higher throughput which translates into better per-
Concurrency control is critical because it is the deciding factor of database performance. In Franaszek, et al. (1992) it is shown that hardware advances have led to several fold improvements in performance and that if the trend continues, CC algorithms will have to be much more efficient just to keep pace with hardware. Indeed, data contention is increasingly the single most critical factor of database performance.

Concurrency control in an object-oriented database is necessarily more complex than it is in other types of databases due to several reasons such as inheritance, the fact that updates to certain parts of the database are not independent from updates to other parts, and the nature of what constitutes an object.

High performance CC mechanisms that produce correct schedules are crucial to the success of any database (Bernstein, et al., 1987). Indeed, any database model will exist only in theory until issues such as concurrency control are examined. The object-oriented database field is emerging as a critical area in database research but work in concurrency control has so far been limited to versioning (Cattell, 1994) and some simple hierarchical locking matrices. A comprehensive theory does not yet exist explaining the objectives, problems, and tradeoffs that must be examined in the area of concurrency control for object-oriented databases.

The paper proceeds with section 2 providing an introduction to concurrency control and object-oriented databases. Section 3 describes the disadvantages and problems of current approaches to object-oriented database concurrency control. Section 4 provides a framework for object-oriented database concurrency control while section 5 includes a description of O^2C^2 detailing the types of locks and when they are used. Section 6 is a summary of the performance analysis described in other research and section 7 concludes with a discussion of the implications of this future research.

Concurrence Control and Object-Oriented Databases

Concurrency Control Mechanisms

Along with coordinating transaction processing in a manner that maintains consistency, it is implicit in concurrency control studies that performance is the other critical factor. That is, all other things equal, mechanisms that support higher levels of performance for a given task are more desirable than mechanisms that support lower levels of performance. In reality, each of the hundreds of CC algorithms provide tradeoffs depending on the characteristics of the application (Bernstein, et al., 1987; Gray & Reuter, 1993; Papadimitriou, 1979). For the purposes of this paper then, concurrency control will be defined as follows:

A concurrency control mechanism allows multiple users to access and update the database so that overall correctness is maintained and performance is optimized. This means that each transaction is executed as though it were processed in isolation (in order to maintain consistency), yet, throughput is maximized.

In the database area, the three standard CC implementations are two-phase locking, timestamping, and optimistic protocols (Bernstein, et al., 1987; Eswaran, et al., 1976; Gray, et al., 1976; Kung & Robinson, 1981). The three CC approaches are briefly reviewed in the following paragraphs. Locking however, was chosen in this research study for two reasons. First, it is the approach most often used for database implementations because it is well understood and can be used for general transactions (Bernstein, et al., 1987; Eswaran, et al., 1976; Gray, et al., 1976). Second, locking schemes are described for a few current object-oriented database implementations including ORION and O^2. These lock-based schemes provide the basis of comparison in a performance study.

Locks are commonly used in order to create a schedule that results in a consistent database. Consistency is determined by comparing the schedules a CC mechanism produces with serial schedules: that is, schedules that only allow one transaction at a time to execute. Serializability is the standard of consistency by which all CC mechanisms are compared. Some basic rules for two-phase locking include (Eswaran, et al., 1976):

1. Whenever a transaction reads or writes a data item, it must hold some kind of lock on that item.
2. At some point before the transaction finishes and becomes inactive, all locks must be released.
3. For any transaction, all lock requests must precede all unlock requests. This means an unlock request is never followed by lock request. If this happens, the transaction must be forced to abort. This is the heart of two-phased locking.
4. A transaction cannot write into the database until after the commit point is reached. This is done to avoid rollbacks (also known as the cascading effect).

These are just the basic rules of two-phased locking. The two-phased protocol is so named because two phases are observed. First, the growing phase accumulates lock requests; second, the shrinking phase unlocks items after processing. Finally, after the transaction reaches the commit point, it is physically written into the database.

The timestamping technique was originally designed for distributed database systems, but many centralized CC mechanisms also utilize some version of timestamps (Bernstein, et al., 1987). The basic rule of timestamp ordering is that older transactions are processed before younger ones. If a younger transaction has already processed a given item and