Semantic Multigranularity Locking for Object-Oriented Database Management Systems

Kyoung-In Kwon
Songchun Moon
Korea Advanced Institute of Science and Technology

Concurrency control schemes for OODBMSs used in the area of performance-critical applications should increase the degree of concurrency and decrease locking overhead in order to offer maximum throughput and minimum response time. However, most commercially available OODBMSs use conventional concurrency control schemes which never exploit the rich semantics of object-oriented data model, so that they fail to offer better performance. In this paper, we propose a concurrency control scheme for OODBMSs, called ISMGL, that not only exploits semantics regarding methods to enhance the concurrency degree, but also utilizes multigranularity locking rules to decrease locking overhead. The novel aspect of ISMGL is that it combines conventional multigranularity locking and nested transaction model with utilizing commutativity of methods. Our concurrency control scheme is applicable for realistic OODBMSs in the presence of complex objects with shared subobjects.

Object-oriented database management systems (OODBMSs) maintain information in terms of objects and classes with providing capabilities for modeling complex objects and the inheritance among classes. An object consists of attributes and methods through which the object’s attributes can be accessed. A class defines both attributes and methods of a group of objects. Methods, furthermore, usually support rich semantics in comparison with simple read or write operations in that the methods represent the behavioral aspects of objects. In general, exploiting semantics regarding these methods are considered to be important to alleviate potential data contention. If the execution order of any two methods never affects the semantic consistency of the execution results, they could obviously execute concurrently. In this respect, if we are able to use semantic information regarding methods, the degree of concurrency could significantly be increased so that OODBMSs are effective enough in the area of performance-critical applications.

Most commercial DBMSs use locking schemes for their concurrency control mechanism due to its simplicity of idea and performance non-inferiority in comparison with any different sort of transaction scheduling schemes so far. In locking schemes, the granule size of lockable unit is considered to have a significant effect on the concurrency degree. By using fine lockable units, in general, the concurrency degree can be enhanced, while the overhead of lock management would be accordingly increased due to the increased number of locks to be managed. To reduce the number of locks which must be acquired to access shared objects, a multigranularity locking (MGL) scheme (Gray, 1978) has been developed by organizing the database in a form of lock instance hierarchy.

Several MGL schemes (Herrmann et al., 1990; Kim, 1990) attempted to schedule concurrent accesses of shared objects by utilizing the notion of class and class composition hierarchies. These schemes took into account the inheritance relationship among classes in the class hierarchy and the aggregation relationship in the class composition hierarchy. However, they were based only on read and write locks. In other words, these schemes lacked features in exploiting the semantic information regarding methods, thus seemed to abandon the possibility of increasing concurrency degree for the performance-critical applications. Example 1 shows this.

Example 1 (Lack of Using Semantics of Methods in MGL):

In a banking database, suppose that a method deposit() is defined in a class account. The deposit(x) method reads the balance of an account object a, say 1,000 dollars, and increases...
it with a certain amount $x$. Note that, the deposit() method is compatible with another deposit() method, since its semantics is to add a given amount to the balance of an account. In Figure 1, therefore, any two deposit() methods belonging to different transactions, say $T_1$ and $T_2$, are allowed to be executed concurrently. Because the two execution sequences <$deposit_1(a,100), deposit_2(a,200)$> and <$deposit_1(a,200), deposit_1(a,100)$> would result in the same 1,300 dollars on the account balance, where <$deposit_1(a,100), deposit_1(a,200)$> means $100$ deposit into the account object $a$ in $T_1$ was processed before $200$ deposit into the same object in $T_2$ was. This means that the banking database would be consistent regardless of the execution sequences.

On the other hand, suppose that $T_1$ and $T_2$ are executing deposit$_1(a,100)$ and deposit$_2(a,200)$ under a conventional MGL scheme. Before anything else, they must set intention exclusive (IX) locks on the banking database and also on the account table, and set an exclusive (X) lock on the record $a$. If $T_1$ sets X lock on a first, $T_2$ cannot acquire X lock on a, since the two X locks obviously conflict each other, as shown in a dotted circle in Figure 1. Therefore, $T_1$ and $T_2$ are not able to be executed simultaneously under the MGL scheme. This means that the MGL misses an opportunity to elevate its concurrency degree.

To exploit the semantics of a method, e.g. deposit() in Example 1, the method should be executed in an atomic way that the method must be executed all or nothing and also executed without interfering against other methods. Otherwise, a concurrent execution of the two deposit() methods could make inconsistent results on the banking database. Therefore, the atomicity of method executions must be ensured by a concurrency control scheme. This can be achieved by treating a method execution as a subtransaction of a top-level transaction. Subtransactions ensure only the atomicity and isolation properties among the well-known ACID properties of transactions: atomicity, consistency, isolation and durability (Haerder & Rothermel, 1993). This means that a subtransaction may commit or abort independently and appears atomic to the other transactions, but the durability of the effects depends on its top-level transaction. Note that a method could invoke other methods so that the calling-called relationship between methods establishes a nested method execution. In this respect, nested method executions can naturally be modeled as nested transactions (Muth et al., 1993; Resende et al., 1994).

Greater degree of concurrency can be achieved by reducing the number of conflicts among methods, while the conflicts are determined by method semantics. The semantics that used for the determination of conflicts are the commutativity of methods. A semantic concurrency control (SCC) scheme (Muth et al., 1993) attempted to exploit the commutativity of methods defined on the same class. In this scheme, methods defined on different classes were presumed to be commutative with each other. However, this might not be true in OODBMSs where complex objects with shared subobjects exist and a subobject could be accessed by methods of more than one objects. The reason is that any two methods defined on different classes could access the same shared subobjects simultaneously with conflicting modes. In this case, the commutativity relationships for all methods defined on different classes should be specified earlier than actual execution of the methods so that a locking scheme can determine the compatibility between methods. Specifying commutativity relationships for all methods in all classes of a database is practically impossible. Note that, in SCC, a method lock is associated with each method execution for the purpose of resolving conflicts between concurrent method executions on the same object.

This sort of commutativity presumption problem in the presence of shared subobjects can be avoided by not associating lock modes with method executions. Instead of controlling conflicts between methods by method locks in accessing shared subobjects, we could take an approach to detect the method-level conflicts via lower-level conflicts between read and write operations at execution time. We call this in-place conflict detection approach in the sense that the conflicts between methods are always detected at the right time instance that the conflicts are posed to take place. The idea is that any conflict between methods that try to access shared subobjects can be detected and controlled instantly when the methods request conflicting locks. In this paper, we propose a locking scheme that takes the in-place conflict detection approach for exploiting method commutativity. The locking scheme also incorporates multigranularity locking rules.

**Related Work and Problems**

Previous work on locking based concurrency control schemes for OODBMSs has concentrated on either the issue...
Related Content

The Expert’s Opinion: A Personal Perspective on the Use of Computing Technology
www.igi-global.com/article/expert-opinion-personal-perspective-use/51204?camid=4v1a

Transaction Concurrency Methods
www.igi-global.com/chapter/transaction-concurrency-methods/11226?camid=4v1a

Customer Investigation Process at Credit Suisse: Meeting the Rising Demands of Regulators
www.igi-global.com/chapter/customer-investigation-process-credit-suisse/8005?camid=4v1a

Annotating Images by Mining Image Search
Xin-Jing Wang, Lei Zhang, Xirong Li and Wei-Ying Ma (2009). Semantic Mining Technologies for Multimedia Databases (pp. 323-349).
www.igi-global.com/chapter/annotating-images-mining-image-search/28840?camid=4v1a