

Minimizing Overhead in Implicit Locking for Object-Oriented Databases

Woochun Jun

Dept. of Computer Education
Seoul National University of Education
Seoul, Korea
wocjun@ns.seoul-e.ac.kr

Abstract. Object-oriented databases (OODBs) have been adopted for many advanced database applications such as artificial intelligence and computer-aided design. In this paper, a locking-based concurrency control scheme for OODBs is presented. The proposed scheme is based on so called the implicit locking and incurs less locking overhead than the implicit locking for any kinds of accesses to OODBs. Especially, the proposed scheme uses only structural information of OODBs so that extra overhead to reduce locking overhead can be minimized. In this work, it is proved that the proposed scheme performs better than the implicit locking.

1 Introduction

OODBs have been widely used for many non-traditional applications such as artificial intelligence and computer-aided design due to advanced modeling power. Among various characteristics in OODBs, the inheritance hierarchy is the one of the major characteristics of an OODB. The inheritance hierarchy enables a subclass to inherit definitions defined on its superclasses. Also, there is an is-a relationship between a subclass and its superclasses. This means that an instance of a subclass is a specialization of its superclasses. In turn, an instance of a superclass is a generalization of its subclasses [5]. Depending on the relationships among classes in OODBs, there are two types of inheritance. In single inheritance, a class can inherit the class definition from one superclass. On the other hand, in multiple inheritance, a class can inherit the class definition from more than one superclass.

In OODBs, there are two access types to an object: instance access (instance read and instance write) and class definition access (class definition read and class definition write) [1]. Also, for access to a class hierarchy, there are two types: class definition writes and query (that is, instance access to all or some instances of a class and its subclasses) [3]. For convenience, these kinds of accesses are denoted as IACH, which mean Instance Access to Class Hierarchy [3,5]. Due to inheritance hierarchy, the definitions of a class' superclasses should not be changed while the class and its instances are being accessed. Also, due to is-a relationship between classes, the search space for a query for a class, says K, may include the instances of all subclasses of the class K as well as all instances of K. For convenience, I call MCA (Multiple Class

Access) for class definition writes and IACHs and SCA (Single Class Access) for other accesses such as class definition read and instance access to a single class.

A concurrency control technique is to coordinate concurrent accesses to the database systems so that it maintains the consistency of the database. A concurrency control scheme allows multiple accesses to a database but may incur an overhead whenever it is invoked. This overhead may degrade the performance of OODBs where many transactions are long-lived. Thus, reducing the overhead is critical to improve overall performance. For OODBs, the implicit locking has been used for controlling accesses [3,4,5,7,8].

This paper is organized as follows. In Section 2, the implicit locking is discussed. In Section 3, a new scheme is proposed. The scheme incurs less locking overhead than the implicit locking does. In Section 4, the correctness of the proposed scheme is proved. Finally, conclusions and future works are described.

2 Implicit Locking

In the implicit locking, in order to set a lock on a class, say K, intention locks are required on each superclass of K along a superclass chain [6]. An intention lock on a class means that some lock is held on a subclass of the class. Then, a lock is required on the class K (called target class). In multiple inheritance, for MCA access, locks are required on the target class and subclasses of the target class, which have more than one superclass. The implicit locking incurs a higher locking overhead when a target class is near the leaf in the class hierarchy due to the intention locks.

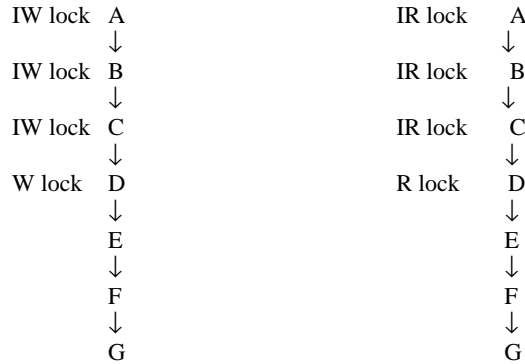


Fig. 1.a. locking for class definition write Fig. 1.b. locking for a query

For example, consider the simple class hierarchy as in Fig. 1. Note that an OODB Orion [3] is selected for the illustration of the implicit locking. For example, in order to update the class definition in class D, the implicit locking works as in Fig. 1.a. For the implicit locking, intention locks IWs corresponding to W (Write) locks are required for all superclasses of D. If another transaction needs to modify the class definition in class, say A, the transaction does not have to check conflicts for each class in the class hierarchy due to the help of the intention lock IW on class A. Since

two lock modes IW and W conflict each other, the incoming lock request is blocked on class A until IW lock is released. In the implicit locking, there is no conflict between intention locks, and between an intention lock and an SCA lock; however, there can be a conflict between an intention lock and an MCA lock depending on the commutativity relationships. Also, locking for a query on D (assuming that all instances of D, E, F and G) is done as in Fig. 1.b.

3 The Proposed Scheme

The basic idea is that, in the implicit locking, some redundant locks can be removed without affecting the correctness of the scheme. In this work, only multiple inheritance is concerned. For single inheritance, the proposed scheme works as in the implicit locking. In the implicit locking, for MCA access type, when a class, say C, is locked, the class C and all subclasses of C which have more than one superclass are locked. In this work, instead of setting locks on all subclasses of C, which have more than one superclass, locks are required on each subclass, say C_i , of C, which has more than one superclass and has only one access path from C to C_i .

Based on the idea above, the proposed scheme is summarized as follows. Assume that a lock is requested on class C. Also, for simplicity, it is assumed that the strict two-phase locking is adopted [6].

Step 1) locking on the root class

For the root class of C, check conflicts and set an intention lock.

Step 2) locking on superclasses

For each superclass along a superclass chain of C, check conflicts and set an intention lock.

Step 3) Locking on the target class

(for SCA access) Check conflicts and set a lock on only the target class C.

(for MCA access) Check conflicts and set a lock for each subclass, C_i , of the target class, which has more than one superclass and only one access path from C to C_i .

Consider the following class hierarchy in Fig. 2. Assume that a class definition needs to be changed in class B. The implicit locking in Orion [3,5] requires getting locks as in Fig. 2.a. On the other hand, locks are required as in Fig. 2.b for the proposed scheme. In the proposed scheme, class H and K needs not be locked since there are at least two access paths from B to those classes. For example, the class H can be reached from the class B using path either B- E – H or B – F – H.

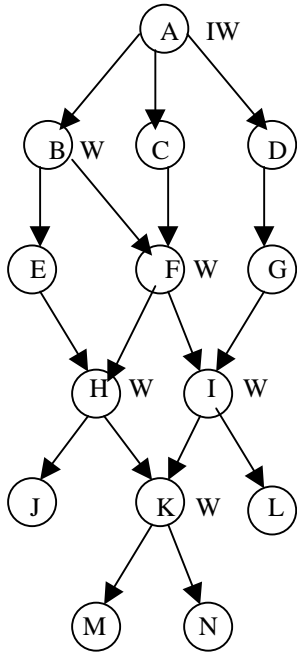


Fig. 2.b. Locks required by the implicit locking

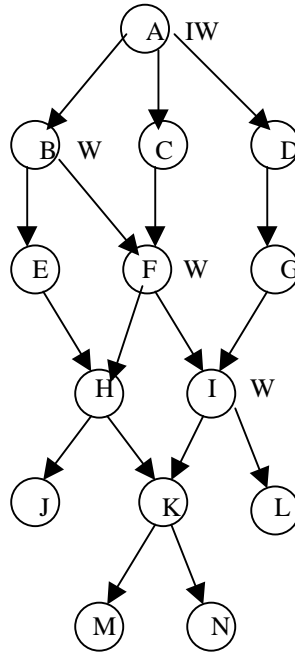


Fig. 2.c. Locks required by the proposed scheme

4 The Correctness of the Proposed Scheme

In this section, I prove that the proposed scheme is correct, that is, it satisfies serializability [2]. Specifically, I prove that, for any requesters, any conflicts with a lock holder are always detected. With this proof, since the proposed scheme is based on two-phase locking, it is guaranteed that the proposed scheme satisfies serializability [2].

Claim) For any lock requester, its conflict with a lock requester (if any) is always detected.

I prove that there exists at least one common class in which both the lock requester and the lock holder can detect a conflict.

Case I) The lock requester and the lock holder have the same target class.

In this case, regardless of intention locks by both requester and holder, the conflict can be detected on the target class.

Case II) The lock requester and the lock holder have the different target class.

Assume that the target classes of the lock holder and the lock requester are C_H and C_R , respectively.

Case II.a) Assume that the class C_H is locked in SCA mode. Then, there are two more cases depending on lock mode by the requester.

1) A lock requester needs an SCA access.

Since it is assumed that the lock requester and the lock holder have the different target classes, there is no conflict.

2) A lock requester needs an MCA access.

If C_R has neither superclass nor subclass relationship with C_H , there is no conflict. Assume that C_R is a subclass of C_H . In this case, there is no conflict. On the other hand, assume that C_R is a superclass of C_H . Then, if C_R is on the same path on which the lock holder sets intention locks, the conflict is detected on C_R . If not, C_H must get a lock on the one of subclasses of C_R , say C_L , which has more than one superclass and has only one access path from C_R to C_L . Otherwise, C_H would not be a subclass of C_R . This means that the possible conflict is detected on C_L .

Case II.b) Assume that the class C_H is locked in MCA mode. Then, there are two more cases depending on lock mode by the requester.

1) A lock requester needs an SCA access.

If C_R is a superclass of C_H , there is no conflict. Also, there is neither superclass nor subclass relationship between C_H and C_R , there is no conflict. Assume that C_R is subclass of C_H . In this case, if C_H is on the same path on which the lock requester sets intention locks, the conflict is detected on the class C_H . If not, the lock requester must get a lock on one of subclasses of C_H , say C_K , which has more than one superclass and has only one access path from C_H to C_K . Otherwise, C_R would not be subclass of C_H . Thus, the conflict will be detected on C_K .

2) A lock requester needs an MCA access.

If C_R has neither superclass nor subclass relationship with C_H , there is no conflict. Assume that C_R is a superclass of C_H . Then, if C_R is on the same path on which the lock holder sets intention locks, the conflict is detected on C_R . If not, C_H must get a lock on the one of subclasses of C_R , say C_L , which has more than one superclass and has only one access path from C_R to C_L . Otherwise, C_H would not be a subclass of C_R . Assume that C_R is a subclass of C_H . In this case, if C_H is on the same path on which the lock requester sets intention locks, the conflict is detected on the class C_H . If not, the lock requester must get a lock on one of subclasses of C_H , say C_M , which has more than one superclass and has only one access path from C_H to C_M . Otherwise, C_R would not be subclass of C_H .

From above, it is concluded that, for any requesters, any conflicts with a lock holder are always detected. This means that the proposed scheme is correct. Since the proposed scheme incurs equal or less locking overhead than the implicit locking, it is proved that the proposed scheme performs better than the implicit locking.

5 Conclusions and Further Works

In this paper, a locking-based concurrency control scheme for OODBs is presented. The proposed scheme is based on the implicit locking scheme and is developed to reduce locking overhead than the implicit locking scheme for any applications. It is proven theoretically that the proposed technique has less locking overhead than the implicit locking does.

Currently I am preparing the simulation work in order to compare the proposed work with the implicit locking. Since most OODBs have long duration transactions, developing a locking-based scheme for such environment will be also good research topic.

References

1. Cart, M. and Ferrie, J., Integrating Concurrency Control into an Object-Oriented Database System, 2nd Int. Conf. on Extending Data Base Technology, Venice, Italy, Mar., (1990), 363 - 377.
2. Eswaran, K., Gray, J., Lorie, R. and Traiger, I., The notion of consistency and predicate locks in a database system, Communication of ACM, Vol. 19, No. 11, Nov., (1976), 624 - 633.
3. Garza, J. and Kim, W., Transaction Management in an Object-Oriented Database System, ACM SIGMOD Int. Conf. on Management of Data, Chicago, Illinois, Jun., (1988), 37 - 45.
4. Jun, W. and Gruenwald, L., An Effective Class Hierarchy Concurrency Control Technique in Object-Oriented Database Systems, Journal of Information And Software Technology, Vol. 40. No. 1, Apr. (1998), 45 - 53.
5. Kim, W., *Introduction to Object-Oriented Databases*, The MIT Press, (1990).
6. Korth, K. and Silberschartz, A., *Database System Concepts*, 2nd Edition, McGraw Hill, (1991).
7. Lee, L. and Liou, R., A Multi-Granularity Locking Model for Concurrency Control in Object-Oriented Database Systems, IEEE Trans. on Knowledge and Data Engineering, Vol. 8, No. 1, Feb., (1996), 144 - 156.
8. Malta, C. and Martinez, J., Controlling Concurrent Accesses in an Object-Oriented Environment, 2nd Int. Symp. on Database Systems for Advanced Applications, Tokyo, Japan, Apr., (1992), 192 - 200.