JOURNAL OF KOREA MULTIMEDIA SOCIETY, VOL. 9, NO. 6, JUNE 2006(pp. 693-699)

Evaluation Metrics for Class Hierarchy in Object-Oriented Databases: Concurrency Control Perspectives

Woochun Jun†

ABSTRACT

Object-oriented databases (OODBs) have been adopted for managing non-standard applications such as computer-aided design (CAD), office document management and many multimedia applications. One of the major characteristics of OODBs is class hierarchy where a subclass is allowed to inherit the definitions defined on its superclasses. In this paper, I present the evaluation metrics for class hierarchy quality in OODBs. These metrics are developed to determine if a concurrency control scheme can achieve good performance or not on a given class hierarchy. I first discuss the existing concurrency control schemes for OODBs. Then I provide evaluation metrics based on structural information and access frequency information in class hierarchies. In order to discuss significance of the proposed performance metrics, an analytical model is developed. Analysis results show that the performance metrics are important factor in concurrency control performance. I consider both single inheritance and multiple inheritance. The proposed metrics can be used to provide guidelines on how to design class hierarchy of an OODB for maximizing the performance of concurrency control technique.

Keywords: Object-oriented database, Class hierarchy, Concurrency Control

1. INTRODUCTION

OODBs have been used for many advanced database applications such as e-commerce with multimedia repositories, document management[13,14]. In a typical OODB, a class object consists of a group of instance objects and class definition objects. The class definition consists of a set of attributes and methods that access attributes of an instance or a set of instances. In OODBs, users can access objects by invoking transactions consisting of a set of method invocations on objects [2].

A concurrency control scheme is used to regulate multiple accesses to a multi-user database so that it maintains database consistency. A concurrency control scheme allows multi-access to a database but incurs an overhead whenever it is invoked. This overhead may degrade the performance of OODBs where many transactions are naturally long-lived. Thus, reducing the concurrency control overhead is crucial to improve the overall performance.

Inheritance is a very important concept in OODBs where a subclass is allowed to inherit the definitions defined on its superclasses. Also, an *is-a* relationship between a subclass and its superclasses is defined so that an instance of a superclass is a generalization of its subclasses [5]. This inheritance relationship between classes forms a class hierarchy. There are two types of accesses to a class hierarchy, MCA (Multiple Class Access) and SCA (Single Class Access) [6]. MCA is an operation accessing possibly more than one class in the class hierarchy. Examples of MCAs include class definition modification operations and instance accesses to all or some instances of a given class and its subclasses. On the other hand, SCA is an operation accessing one class in the hierarchy.

※ Corresponding Author : Woochun Jun, Address : (137-742) 1650 Seocho-Dong, Seocho-Gu, Seoul, Korea, TEL : +82-2-3475-2504, FAX : +82-2-3475-2263, E-mail : wocjun@snue.ac.kr
Receipt date : Feb. 16, 2006, Approval date : June. 8, 2006
† Dept. of Computer Education, Seoul National University of Education

Examples of SCAs are class definition read operations and instance access to a single class. For a lock-based concurrency control scheme, when an MCA operation is requested on some class C, it may be necessary to get locks for C as well as all subclasses of C.

In this paper, I provide evaluation metrics for class hierarchy quality in OODBs. These metrics are developed to determine if a concurrency control scheme can achieve good performance or not on a class hierarchy. The typical performance metrics of a concurrency control scheme are response time and locking overhead (usually represented as the number of locks needed for an access) [1,10]. Especially, the proposed metrics for class hierarchy can be used to determine if a concurrency control scheme can incur less locking overhead or not. The proposed evaluation metrics are based on both structural information and access frequency information in class hierarchy. I also provide evaluation metrics for both single inheritance and multiple inheritance.

2. EXISTING CONCURRENCY CONTROL SCHEMES

2.1 Concurrency Control Schemes Based on Structural Information

2.1.1 Single Inheritance

In the literature, there are two major locking-based approaches dealing with a class hierarchy: explicit locking [2,12] and implicit locking [5,9,10,11]. In explicit locking, for an MCA operation on a class, C, a lock is set not only on the class C, but also on each subclass of C in the class hierarchy. For an SCA operation, a lock is set for only the class to be accessed (also called target class). Thus, for an MCA, transactions accessing a class near the leaf in a class hierarchy will require fewer locks than transactions accessing a class near the root in the class hierarchy. Also, explicit locking can treat in the same way for both single inheritance, where a class can inherit the class definition from one superclass, and multiple inheritance where a class can inherit the class definition from more than one superclass. But, explicit locking requires more locking overhead for transactions accessing a class near the root in a class hierarchy.

Implicit locking is based on intention locks [3]. An intention lock on a class indicates that some lock is set on a subclass of the class. Thus, when a lock is set on a class C, extra locks are required to be set on a path from C to its root as well as on C. In implicit locking, when an MCA operation is accessed on a class C, locks are not required for every subclass of the class C. It is sufficient to set a lock only on the class itself. Thus, for an MCA access, it incurs less locking overhead than explicit locking. But, implicit locking requires more locking overhead when a target class is near the leaf class in a class hierarchy due to intention lock overhead.

2.1.2 Multiple Inheritance

In explicit locking, no special steps are not required for dealing with multiple inheritance. But, in implicit locking, for the MCA access type, when a class C is locked, all subclasses of C which have more than one superclass are also locked [5,9]. As an example, consider the following simple class hierarchy shown in Fig. 1. Note that the lock modes are based on an OODB system called Orion[5]. In order to modify the class definition in class, say F, the explicit locking scheme works as in Fig. 1.a. The W (Write) locks are required for each subclass of F as well as the target class F. On the other hand, for the implicit locking scheme, intention locks IWs corresponding to W locks are required for each superclass on the path from F to the root A. Assume that path A-C-F is selected. Also, classes I and J, which are subclasses of F, are locked since those classes have more than one superclass. Fig. 1.b shows locks required by the implicit locking scheme.

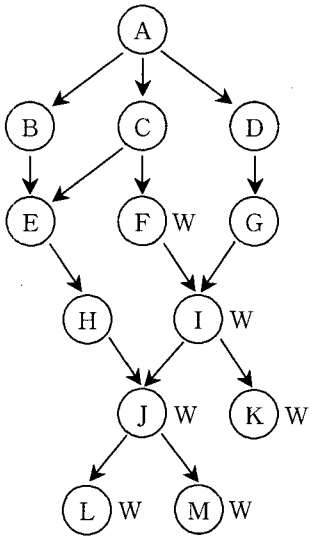

Fig. 1a. Locks by explicit locking.

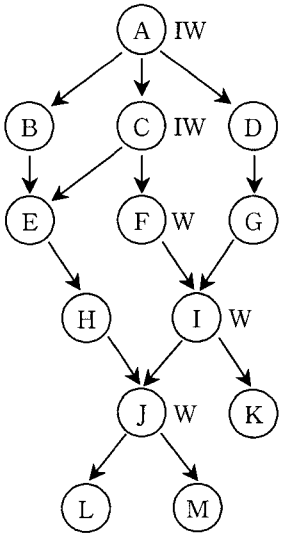

Fig.1b. Locks by implicit locking.

2.2 Concurrency Control Schemes Based on Access Frequency Information

In [6], the SC-based concurrency control scheme is proposed to provide better system performance than the existing schemes, explicit locking and implicit locking. The SC-based scheme is based on the concept of Special Class (SC) where an SC is a class on which MCA operations are performed frequently. In this scheme, intention locks are set on only SCs. Thus, locking overhead is less than that of implicit locking which requires intention locks to be set on every superclass of the target class. Also, in order to have less locking overhead than explicit locking, the following principle is adopted: for an SCA access, a lock is set on only the target class like in explicit locking. For an MCA access, unlike explicit locking, locks are set on every class from the target class to the first SC through the subclass chain of the target class. If there is no such SC, then locks are set on leaf classes. If the target class is an SC itself, then a lock is set only on the target class.

The scheme is summarized as follows. Assume that a lock is requested on class C. For simplicity, strict two-phase locking [1,4] is adopted.

Step 1) Locking on SCs

- For each SC (if any) through the superclass chain of C, check conflicts and set an intention lock.

Step 2) Locking on a target class

- If the lock request is an SCA, check conflicts with locks set by other transactions and set a lock on only the target class C and set a lock on an instance of C if a method is invoked on the instance
- If the lock request is an MCA, then, from class C to the first *SC* (or leaf class if there is no SC) through the subclass chain of C, check conflicts and set a lock on each class. If class C is an SC, then set a lock only on C.
- If class C has more than one subclass, perform the same step (2) for each subclass chain of C.

For the SC-based scheme, the following SC assignment scheme is adopted [6]. Assuming that the number of accesses to each class is stable and the access frequency (of MCA and SCA) to each class is known in advance, the SC assignment scheme is constructed as follows.

//Start from each leaf class until all classes are checked //

Step 1) If a class is a leaf, then the class is assigned as non-SC.

If a class C has not been assigned yet and all subclasses of C have been already assigned, then do the following:

for class C and all of its subclasses,

calculate the number of locks (N_1) when the class is assigned as SC

calculate the number of locks (N_2) when the class is assigned as non-SC

Step 2) Assign it as SC only if $N_1 < N_2$

In [7], a concurrency control scheme is presented. The scheme is based on the SC-scheme but improves it as follows. The basic idea is that some redundant locks can be reduced without affecting the correctness of the scheme. Assume that a class C is accessed and thus it needs to be locked. For the SC-based scheme, an intention lock is set on every SC through the superclass chain of C. On the other hand, the proposed scheme does not have to set intention locks on every SC through the superclass chain. That is, only the first SC near the root and the last SC near class C need to be locked as long as SCs excluding the first SC and the last SC have only one subclass. The detailed concurrency control scheme and the proof of its superiority over the SC-scheme are shown in [7].

For example, consider the class hierarchy as in Fig. 2.a. Also, assume that locks are requested by T_1 and T_2 as follows.

1) T_1: class definition update operation on class C6

2) T_2: class definition update operation on class C7

As in Fig 2.a, 2.b, 2.c and 2.d, 6, 7, 9 and 13 locks are required for T_1 and T_2 by the proposed scheme in [7], SC-based scheme, explicit locking, and implicit locking, respectively.

3. EVALUATION METRICS FOR CLASS HIERARCHY

In this section I provide evaluation metrics for class hierarchy quality. These metrics for class hierarchy can be used to determine if a concurrency control scheme can incur less locking overhead or not.

3.1 Evaluation Metrics based on Structural Information

3.1.1 Single inheritance

As discussed in the previous section, in single inheritance, the number of locks required for an access to class, say C, are in proportion to the number of subclasses and the number of superclasses of C.

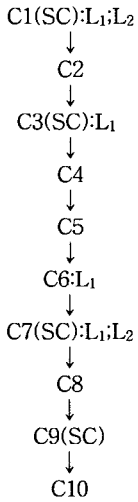

Fig 2a. Locks by Scheme in [7].

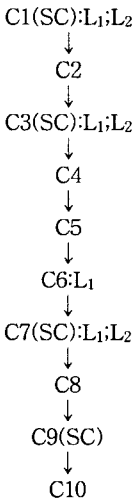

Fig. 2b. Locks by SC-based scheme.

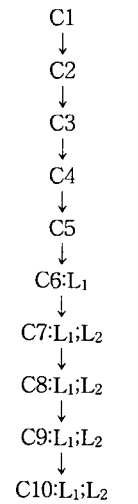

Fig. 2c. Locks by Explicit locking.

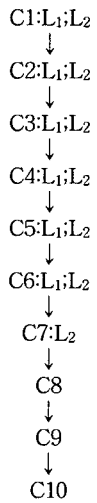

Fig. 2d. Locks by Implicit locking.

Below I define the average number of subclasses for each class in a given class hierarchy, NUM_{SUB}. This metric is used in evaluating explicit locking.

$$NUM_{SUB} = (SUB_1 + SUB_2 + \cdots + SUB_N) / N$$

where SUB_I is the number of subclasses for class I and N is the total number of classes in a given class hierarchy

Below I also define the average number of superclasses for each class in a given class hierarchy, NUM_{SUP}. This metric is helpful in evaluating implicit locking:

$$NUM_{SUP} = (SUP_1 + SUP_2 + \cdots + SUP_N) / N$$

where SUP_I is the number of superclasses for class I and N is the total number of superclasses in a given class hierarchy

3.1.2 Multiple Inheritance

For multiple inheritance, in explicit locking, no evaluation metrics are necessary since this scheme treats both single inheritance and multiple inheritance in the same way. However, in implicit locking, when a MCA lock is required for class C, all subclasses of C which have more than one superclass need to be locked as well. Thus, for implicit locking, I define $NUM_{SUB\text{-}SUP}$ to be the average number of subclasses having more than one superclass in a class hierarchy as follows:

$$NUM_{SUB\text{-}SUP} = (SUB\text{-}SUP_1 + SUB\text{-}SUP_2 + \cdots + SUB\text{-}SUP_N)/N$$

where $SUB\text{-}SUP_I$ is the number of subclasses having more than one superclass for class I and N is the total number of classes in a given class hierarchy

3.2 Evaluation Metrics Based on Access Frequency Information

The two concurrency control schemes based on access frequency information that have been proposed in [6, 7] require accurate information on access frequency for each class. Otherwise, either explicit locking or implicit locking should be applied depending on the access types of transactions. That is, if transactions accessing a class near the root in a class hierarchy are dominant, implicit locking is a better choice. Otherwise, explicit locking outperforms the implicit locking.

If the access information on a class hierarchy is available, it should be stable. Otherwise, the SC assignment needs to be done frequently. For each class, I can define two evaluation metrics as follows: the access frequency on class I by transactions requesting an SAC type of access, $ACC\text{-}FREQ_{SAC}$ (I), and the access frequency on class I by transactions requesting an MAC type of access, $ACC\text{-}FREQ_{MAC}$ (I).

4. SIGNIFICANCE OF THE PROPOSED EVALUATION METRICS

The proposed evaluation metrics are developed to reflect number of locks by various concurrency control schemes for a given OODB access. In order to show that the proposed evaluation metrics are significant, it will be shown that number of locks is a significant factor in concurrency control performance. Especially, I will discuss that number of locks is significant in transaction response time.

For this purpose, a simple analytical model is developed as follows. In this model, lock contention probability and lock waiting time are not considered. This model is to compare response time only for varying number of locks to a given database access.

$$RES = E + N*(R_{SET} + R_{REL})$$

Where RES is mean response time, E is sum of execution time for N granules, R_{SET} is mean time to set a lock by a transaction, R_{REL} is mean time to release lock by a transaction. For simplicity, I assume that response time consists of only sum of execution time for database accesses (N granules) and sum of time to set and release locks for N granules. Note that response time is a major standard for concurrency control performance in databases.

Table 1 shows analytical parameters adopted from literature [8]. More general analytical model can be found in [15].

Table 1. Analytical Parameters

Parameters	Value
R_{RES-I} (Mean time to set a lock by an instance access transaction)	0.3641 ms
R_{RES-C} (Mean time to set a lock by an class definition access transaction)	0.3522 ms
R_{REL-I} (Mean time to release a lock by instance access transaction)	0.0035 ms
R_{REL-C}(Mean time to release a lock by class definition access transaction)	0.0011 ms
E (Execution time for each granule)	2 ms

If the above model and parameters are applied to class hierarchies in Fig. 2 (single inheritance) and Fig. 1 (Multiple inheritance), respectively, response time can be obtained as in Table 2.

Note that response time for single inheritance is obtained from two transactions while response time for multiple inheritance is obtained from one transaction. In multiple inheritance, reducing one lock results in response time enhancement by 10%.

Results obtained from Table 2 shows that reducing number of locks are significant factors in response time. It means that reducing number of locks can improve concurrency control performance. In turn, reducing class hierarchy depth is a major factor to reduce number of locks required.

For the same number of instance objects, class hierarchy with low depth can increase number of instance objects for each class object. This means that, for such class hierarchy, locks on class objects can degrade concurrency. This is due to that more instance objects are blocked for a lock. Thus, class hierarchy with low depth can reduce number of locks but may degrade concurrency among concurrent transactions since low concurrency may cause high lock contention.

5. CONCLUSIONS AND FURTHER WORK

In this paper, I presented the evaluation metrics for the quality of class hierarchies in OODBs. These metrics is to determine if a concurrency control scheme can achieve good performance or not on a class hierarchy in terms of locking overhead. In the evaluation metrics, both single inheritance and multiple inheritance are considered. Also, the proposed evaluation metrics are developed based on the structural information and access frequency information on class hierarchies.

In order for a concurrency control scheme to have less locking overhead, it is important to maintain the followings for a class hierarchy: 1) low average number of subclasses or superclass for each class 2) low average number of subclasses having more than one superclass 3) the accurate and stable access frequency information. However, having low class hierarchy depth for less locking overhead may degrade concurrency among transactions so that overall performance may be degraded.

Currently I am developing a concurrency control scheme for controlling access to composite object hierarchies, which is also a major characteristic in

Table 2. Response time of various concurrency control schemes

Inheritance type	Concurrency control	Response time
Single inheritance	Improved SC-based locking [7]	6.1198 ms
	SC-based locking	6.4731 ms
	Explicit locking	7.1797 ms
	Implicit locking	8.5929 ms
Multiple inheritance	Explicit locking	4.1198 ms
	Implicit locking	3.7665 ms

OODBs. It is interesting to develop the evaluation metrics for composite object hierarchies.

6. REFERENCES

[1] P. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recoveryin Database Systems*, Reading, Massachusetts, Addison-Wesley, 1987.

[2] M. Cart and J. Ferrie, "Integrating Concurrency Control into an Object-Oriented Database System," *2nd Int. Conf. on Extending Data Base Technology*, Venice, Italy, pp. 363-377, Mar., 1990.

[3] C. Date, *An Introduction to Database Systems*, Vol. II, Reading, Massachusetts, Addison-Wesley, 1985.

[4] K. Eswaran, J. Gray, R. Lorie and I. Traiger, "The Notion of Consistency and Predicate Locks in a Database System," *Communication of ACM*, Vol. 19, No. 11, pp. 624-633, Nov., 1976.

[5] J. Garza and W. Kim, "Transaction Management in an Object-Oriented Database System," *ACM SIGMOD Int. Conf. on Management of Data*, Chicago, Illinois, pp. 37-45, Jun. 1988.

[6] W. Jun and L. Gruenwald, "An Effective Class Hierarchy Concurrency Control Technique in Object-Oriented Database Systems," *Journal of Information And Software Technology*, Vol. 40. No. 1, pp. 45-53, Apr., 1998.

[7] W. Jun and L. Gruenwald, "An Optimal Locking Scheme in Object-oriented Database Systems," *The First International Conference on Web-Age Information Management*, Shanghai, China, pp. 95-105, Jun., 2000.

[8] W. Jun, "A Multi-granularity locking-based concurrency control in object-oriented database systems," *The Journal of Systems and Software*, Vol. 54, pp. 210-217, Nov., 2000.

[9] W. Jun and S. Hong, "Controlling Concurrent Accesses in Multimedia Databases for Decision Support," *5th Pacific Rim Conference on Multimedia*, Tokyo, Japan, pp. 180-187, Nov./Dec., 2004.

[10] L. Lee and R. Liou, "A Multi-Granularity Locking Model for Concurrency Control in Object-Oriented Database Systems," *IEEE Transaction on Knowledge and Data Engineering*, Vol. 8, No. 1, pp. 144-156, Feb., 1996.

[11] C. Malta and J. Martinez, "Controlling Concurrent Accesses in an Object-Oriented Environment," *2nd Int. Symp. on Database Systems for Advanced Applications*, Tokyo, Japan, pp. 192-200, Apr., 1992.

[12] C. Malta and J. Martinez, "Automating Fine Concurrency Control in Object-Oriented Databases," *9th IEEE Conf. on Data Engineering*, Vienna, Austria, pp. 253-260, Apr., 1993.

[13] M. O'Docherty, *Object-oriented Analysis and Design*, John Wiley & Sons Inc., 2005.

[14] S. Schach, *Object-oriented and Classical Software Engineering* 6/E, McGraw Hill Press, 2004.

[15] P. Yu, D. Dias, J. Robinson, B. Iyer and D. Cornell, "Modelling of Centralized Concrrency Control in a Multi-System Environment," *Performance Evaluation Review*, Vol. 13, No. 2, pp. 183-191, 1985.

Woochun Jun

1985 Dept. of Computer Science, Sogang University (B.S.)

1987 Dept. of Computer Science, Sogang University (M.S.)

1997 School of Computer Science, University of Oklahoma, USA (Ph.D.)

1998~Present : Associate Professor, Dept. of Computer Education, Seoul National University of Education

Research areas : Database, Web-based Learning, Mobile Learning