J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

Development of Performance Evaluation Metrics of Concurrency Control in Object-Oriented Database Systems☆

Woochun Jun[1] Suk-Ki Hong[2*]

ABSTRACT

Object-oriented databases (OODBs) canbe used for many non-traditional database application areas such as computer-aided design, etc. Usually those application areas require advanced modeling power for expressing complicated relationships among data sets. OODBs have more distinguished features than the traditional relational database systems. One of the distinguished characteristics of OODBs is class hierarchy (also called inheritance hierarchy). A class hierarchy in an OODB means that a class can hand down the definitions of the class to the subclass of the class. In other words, a class is allowed to inherit the definitions of the class from the superclass. In this paper, we present performance evaluation metrics for class hierarchy in OODBs from a concurrency control perspective. The proposed performance metrics are developed to determine which concurrency control scheme in OODBs can be used for a given class hierarchy. In this study, in order to develop performance metrics, we use class hierarchy structure (both of single inheritance and multiple inheritance), and data access frequency for each class. The proposed performance metrics will be also used to compare performance evaluation for various concurrency control techniques.

☞ keyword : Object-oriented Database, Concurrency Control, Performance Evaluation Metrics, Class Hierarchy, Class Composition Hierarchy

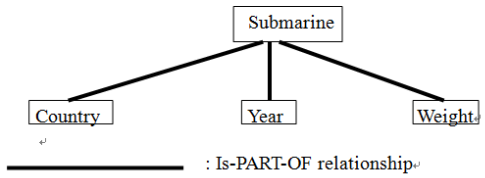
1. Introduction

OODBs were adopted in advanced database areas like CAD (Computer-Aided Design), CASE (Computer-Aided Software Engineering), and so on. For those advanced database applications, the traditional relational database systems are not sufficient for modeling those advanced applications. This is because the relationships among various entities are too complicated to express in table format in a relational database system.

An OODB can provide complex modeling power. To put it concretely, a typical OODB can provide two advanced relationships. The first relationship is class hierarchy. Class hierarchy is also called IS-A hierarchy in which a class object is made up of the groups of instance objects and class definition objects. Class hierarchy is concerned with inheritance among objects. That is, a subclass of a class, say A, inherits definitions of the class A. The second relationship in OODB is IS-PART-OF hierarchy,also called composite object hierarchy. In a composite object hierarchy, for example, in Figure 1, a submarine object has country, year and weight as composite objects.

In OODBs, a transaction is usually composed of a set of methods that are used to retrieve and change values of an object [1]. Usually, in a database system, there are many concurrently running transactions. These concurrently running transactions may violate database consistency. For this reason, database concurrency control technique is required to maintain consistency among transactions at any time. However, while a concurrency control technique is maintaining consistency, it may incur various overhead on database. This overhead may degrade overall database performance. Especially a transaction in an OODB is usually long-lived so that database performance may be degraded, resulting overall transaction response time damaged.

1 Dept. of Computer Education, Seoul National University of Education, Seoul, 06639, Korea
2 Dept. of Business Administration, Dankook University, Yongin, 16890, Korea
* Corresponding author(skhong017@dankook.ac.kr)
[Received 20 October2017, Reviewed 6 November 2017(R2 14 March 2018, R3 25 April 2018), Accepted 15 May 2018]
☆ This paper is an extended version of the paper presented and published in APIC-IST 2017 conference.



(Fig. 1) IS-PART-OF hierarchy

Inheritance is a key concept in OODBs. Two types of accesses for class hierarchy are MCR (Many-Class Retrieve) and OCR (One-Class Retrieve) [2]. MCR is the type of access request to possibly more than one class. Examples of MCR include class definition update and a query (Instance accesses to all or some instances of a given class and its subclasses). On the other hand, SCR is a type of access request to only one class. Possible SCRs include the class object retrieve, and the instance object access to only one class.

In this paper, we provided performance evaluation metrics concurrency control techniques for class hierarchy in OODBs. The proposed evaluation metrics can be used to determine what kind of concurrency control technique is used for a given class hierarchy. The typical performance evaluation metrics of a database concurrency control scheme are response time and locking overhead [3,4]. The proposed performance evaluation metrics are based on structural information and data access frequency for each class in class hierarchy. Further, we provided evaluation metrics for both of multiple inheritances and single inheritance. In Section 2, we discuss relatedworks on the existing concurrency control techniques. In Section 3, we developed the performance evaluation metrics for concurrency control schemes dealing with class hierarchy. Finally, in Section 4, we present conclusions and further research issues.

2. Related Works

2.1 Single Inheritance

According to the previous works, there are two representative locking-based works for a class hierarchy: explicit locking [1,5] and implicit locking [4,6,7,8], respectively.

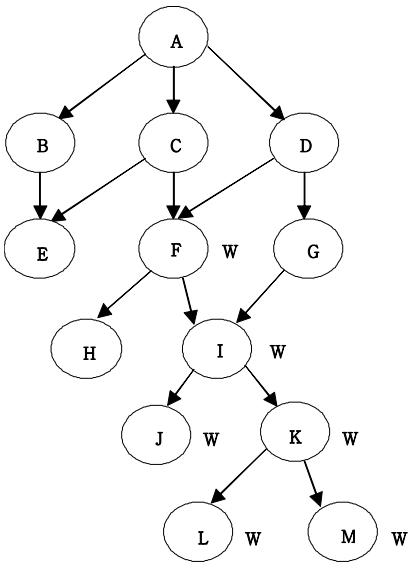
In implicit locking, locking on a class, say A, causes intention locks on all superclasses of the class A. It means that every superclass of A needs an intention lock. An intention lock on a class, say A, indicates that there is an actual lock on some subclass of A. An intention lock is used to detect possible conflict in advance.

In explicit locking, for an MCR access on a class C, a lock is required on the class C as well as each subclass of C. In the meanwhile, for an SCR request, a lock is required on only the class C. Therefore, for an MCR request, if a transaction retrieves a class in the bottom of a class hierarchy, the transaction will need less number of locks than the transactions retrieving a class in the top of class hierarchy. Moreover, explicit locking can do exactly the same way for both single inheritance and multiple inheritances. But, explicit locking needs more number of locks for transactions retrieving a class in the top of a class hierarchy.

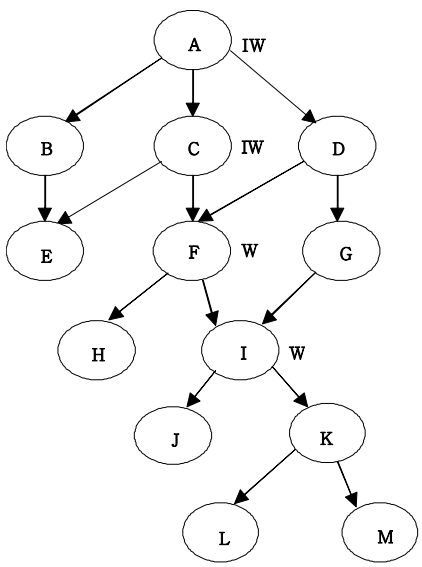
2.2 Multiple Inheritance

Explicit locking gets locks exactly same as single inheritance. It means that explicit locking scheme does not have to take any different actions for multiple inheritances.

However, in implicit locking, for the MCR access to a class C. locks are needed for both the class C and all subclasses of C that have more than one superclass [2,6]. For instance, take a look at the following class hierarchy with multiple inheritances as shown in Figure 2. In figure 2, the lock modes are applied on an Orion OODB [6]. In this class hierarchy, in order to change the class definition, for instance F, the explicit locking scheme does as shown in Figure 2. The W (Write) locks are needed for all subclasses of F as well as the class F. Meanwhile, in the implicit locking scheme, intention locks IWs corresponding to W locks are needed for each superclass of F and also F. Among two paths from F to the root class A, A-C-F and A-D-F, assume that path A-C-F is selected. In addition, class I is also locked are locked since the class I has more than one superclass. Figure 3 shows locks by the implicit locking scheme.



(Fig. 2) Locks by explicit locking



(Fig. 3) Locks by implicit locking

3. Development of Performance Evaluation

In this section we provide performance evaluation metrics for concurrency control schemes dealing with class hierarchy. For given class hierarchy, the proposed metrics represents possible lock requirement. The proposed metrics can be used to check if a concurrency control scheme can incur less locking overhead for a given class hierarchy. In other words, for a given class hierarchy, those metrics are used to determine what kind of concurrency control schemes dealing with class hierarchy is used.

3.1 Single Inheritance

Depending on whether a concurrency control is based on implicit locking or explicit locking, the number of locks required for an access to a class C may require locks on subclasses of C and/or locks on superclasses of C.

In implicit locking, locking on C means that locks are also required on each superclass of C as well. That is, intention locks are required on each superclass of C. On the other hand, in explicit locking, locking on C means that locks are also required on each subclass of C depending on lock request types. For class definition read for instance read on class C, locks on each subclass of C are not required. However, for class definition change and query on class C, locks on each subclass of C are necessary.

We developed two performance metrics for class hierarchywith single inheritance. First of all, we assume that data access frequency on each class in a given class hierarchy is known in advance.

Assuming that data access frequency for a class C is represented as F_C, all locks required for a given class C,

$$LOCKS\_ON\_SUPERCLASS_C = F_1 + F_2 + F_3 + \text{,,,,,,,,} F_{C-1}$$

Where $LOCKS\_ON\_SUPERCLAS_C$ is all intention locks required for a given class C, and also 1,2,,,,C-1 are all of superclasses of C.

On the other hand, for explicit locking, all locks required for a given class C can be calculated as follows.

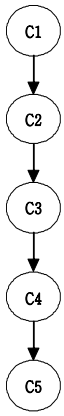
$$LOCKS\_ON\_SUBCLASS_C = P_{C\text{-}MCR} *(F_{C+1} + \text{,,,,,,,} + F_{N-1} + F_N)$$

Where $LOCKS\_ON\_SUBCLASS_C$ is all locks required for C+1, C+2⋯.N-1, N that are all subclasses of C. $P_{C\text{-}MCR}$ is a probability that a class C is accessed by MCR access

type. $P_{C\text{-}SCR}$ is a probability that a class C is accessed by SCR access type.

$$P_{C\text{-}SCR} + P_{C\text{-}MCR} = 1$$

For example, consider the following class hierarchy in Figure 4 and data access frequency for each class is given as in Table 1.



(Fig. 4) A Class Hierarchy

(Table 1) Data Access Frequency and Probability of MCR /SCR

Class	Data Access Frequency	Prob. Of MCR/SCR
C1	50	0.3/0.7
C2	40	0.2/0.8
C3	10	0.1/0.9
C4	20	0.1/0.9

Based ondata access frequency on each class and probability of MCR, LOCKS_ON_SUPERCLASS of each class can be calculated as follows.

$LOCKS\_ON\_SUPERCLASS_{C1} = 0$
$LOCKS\_ON\_SUPERCLASS_{C2} = 40$
$LOCKS\_ON\_SUPERCLASS_{C3} = 20$
$LOCKS\_ON\_SUPERCLASS_{C4} = 60$
$LOCKS\_ON\_SUPERCLASS_{C5} = 80$

On the other hand, LOCKS_ON_SUBCLASS of each class can be calculated as follows.

$LOCKS\_ON\_SUBCLASS_{C1} = 0.3 * 50 * 4 = 60$
$LOCKS\_ON\_SUBCLASS_{C2} = 0.2 * 40 * 3 = 24$
$LOCKS\_ON\_SUBCLASS_{C3} = 0.1 * 10 * 2 = 2$
$LOCKS\_ON\_SUBCLASS_{C4} = 0.1 * 20 * 1 = 2$
$LOCKS\_ON\_SUBCLASS_{C5} = 0$

For class C1, locks on subclasses are required for only an MCR access. Thus, $LOCKS\_ON\_SUBCLASS_{C1}$ is calculated as follows.

$LOCKS\_ON\_SUBCLASS_{C1}$ =
0.3 * 50 (locks for C2)
+ 0.3*50(locks for C3)
+ 0.3*50(locks for C4)
+ 0.3*50(locks for C5)
= 60

Likewise, $LOCKS\_ON\_SUBCLASS_{C2}$, LOCKS_ON_ $SUBCLASS_{C3}$, $LOCKS\_ON\_SUBCLASS_{C4}$ can be calculated in the same way. For class C5, $LOCKS\_ON\_SUBCLASS_{C5} = 0$ since the class C5 is a leaf class.

Finally, we develop LOCKS_ON_ SUPERCLASS and LOCKS_ON_SUBCLASS as follows.

$LOCKS\_ON\_SUPERCLASS$ =
$LOCKS\_ON\_SUPERCLASS_1$
+ $LOCKS\_ON\_SUPERCLASS_2$
+
,,,,,,,,
+$LOCKS\_ON\_SUPERCLASS_{N-1}$
+$LOCKS\_ON\_SUPERCLASS_N$

$LOCKS\_ON\_SUBCLASS = LOCKS\_ON\_SUBCLASS_1$+
$LOCKS\_ON\_SUBCLASS_2$
+
,,,,,,,,
+$LOCKS\_ON\_SUBCLASS_{N-1}$
+$LOCKS\_ON\_SUBCLASS_N$

That is, LOCKS_ON_SUPERCLASS and LOCKS_ON_ SUBCLASS represent total locks required for intention locks (for implicit locking) and subclass locking (for explicit

locking), respectively. Using a class hierarchy in Figure 4 and data access frequency in Table 1, LOCKS_ON_ SUPERCLASS and LOCKS_ON_SUBCLASS are calculated as follows.

LOCKS_ON_SUPERCLASS = 0 + 40 + 20 + 60 + 80 = 200
LOCKS_ON_SUBCLASS = 60 + 24 + 2 + 2 + 0 = 88

In this example, since LOCKS_ON_ SUPERCLASS > LOCKS_ON _SUBCLASS, it is better to use explicit locking rather than implicit locking. Two performance evaluation metrics, LOCKS_ON_SUPERCLASS and LOCKS_ON_SUBCLASS are used to determine which concurrency control schemes, implicit locking and explicit locking, is better to reduce locking overhead for a given class hierarchy.

3.2 Multiple Inheritance

As discussed earlier, for multiple inheritance, explicit locking works exactly same as single inheritance. However, in implicit locking, for an MCR lock in class C, all subclasses of C that have more than one superclass need to be also locked. Thus, for implicit locking, LOCKS_ON_ $SUBCLASS_C$ (Implicit) is defined as follows.

$$LOCKS\_ON\_SUBCLASS_C\ (Implicit) = P_{MCR} * (F_A + F_B + ,,,,,,, + F_{K-1} + F_K)$$

Where F_A, F_B,,,,,,F_{K-1}, and F_K are subclasses of the class C that have more than one superclass.

For entire classes in a class hierarchy, LOCKS_ON_ SUBCLASS (Implicit) is defined as overall locks required for all classes that have more than one super class.

LOCKS_ON_SUBCLASS (Implicit) =
$LOCKS\_ON\_SUBCLASS_1$ (Implicit)
+ $LOCKS\_ON\_SUBCLASS_2$ (Implicit)
+
,,,,,,,,
+$LOCKS\_ON\_SUBCLASS_{N-1}$(Implicit)
+$LOCKS\_ON\_SUBCLASS_N$ (Implicit)

For implicit locking, in multiple inheritance, total number of locks for implicit locking, LOCKS_ON_IMPLICIT, is defined as follows.

LOCKS_ON_IMPLICIT = LOCKS_ON_SUPERCLASS + LOCKS_ON_SUBCLASS (Implicit)

In the meanwhile, total number of locks for explicit locking, LOCKS_ON_EXPLICIT, is defined as follows.

LOCKS_ON_EXPLICIT= LOCKS_ON_SUBCLASS

For a class hierarchy in Figure 2, assume that data access frequency for each class is given in Table 2.

(Table 2) Data Access Frequency and Probability of MCR /SCR

Class	Data Access Frequency	Prob. Of MCR/SCR
A	50	0.2/0.8
B	40	0.1/0.9
C	20	0.2/0.8
D	50	0.1/0.9
E	40	0.1/0.9
F	20	0.3/0.7
G	40	0.1/0.9
H	60	0.1/0.9
I	30	0.2/0.8
J	20	0.1/0.9
K	80	0.1/0.9
L	80	0.2/0.8
M	60	0.1/0.9

As in single inheritance, LOCKS_ON _SUPERCLASS for each class is calculated as follows. For simplicity, assume that, for a class that has more than one superclass, only leftmost superclass is selected for intention locks. For example, for class E, intention locks are set on class B and A rather than C and A.

$LOCKS\_ON\_SUPERCLASS_A = 0$
$LOCKS\_ON\_SUPERCLASS_B = 40$
$LOCKS\_ON\_SUPERCLASS_C = 20$
$LOCKS\_ON\_SUPERCLASS_D = 50$

$LOCKS\_ON\_SUPERCLASS_E = 80$
$LOCKS\_ON\_SUPERCLASS_F = 40$
$LOCKS\_ON\_SUPERCLASS_G = 80$
$LOCKS\_ON\_SUPERCLASS_H = 180$
$LOCKS\_ON\_SUPERCLASS_I = 90$
$LOCKS\_ON\_SUPERCLASS_J = 80$
$LOCKS\_ON\_SUPERCLASS_K = 320$
$LOCKS\_ON\_SUPERCLASS_L = 400$
$LOCKS\_ON\_SUPERCLASS_M = 300$

On the other hand, LOCKS_ON_ SUBCLASS for each class is calculated as follows.

$LOCKS\_ON\_SUBCLASS_A = 0.2 * 50 * 12 = 120$
$LOCKS\_ON\_SUBCLASS_B = 0.1 * 40 = 4$
$LOCKS\_ON\_SUBCLASS_C = 0.2* 20 * 8 = 32$
$LOCKS\_ON\_SUBCLASS_D = 0.1 * 50 * 8 = 40$
$LOCKS\_ON\_SUBCLASS_E = 0$
$LOCKS\_ON\_SUBCLASS_F = 0.3 * 20 * 6 = 36$
$LOCKS\_ON\_SUBCLASS_G = 0.1 * 40 * 5 = 20$
$LOCKS\_ON\_SUBCLASS_H = 0$
$LOCKS\_ON\_SUBCLASS_I = 0.2 * 30 * 4 = 24$
$LOCKS\_ON\_SUBCLASS_J = 0$
$LOCKS\_ON\_SUBCLASS_K = 0.1 * 80 * 2 = 16$
$LOCKS\_ON\_SUBCLASS_L = 0$
$LOCKS\_ON\_SUBCLASS_M = 0$

Likewise, LOCKS_ON_SUBCLASS (Implicit) for each class is calculated as follows.

$LOCKS\_ON\_SUBCLASS_A$ (Implicit) $= 0.2 * 50 * 3 = 30$
$LOCKS\_ON\_SUBCLASS_B$ (Implicit) $= 0$
$LOCKS\_ON\_SUBCLASS_C$ (Implicit) $= 0.2* 20 * 2 = 8$
$LOCKS\_ON\_SUBCLASS_D$ (Implicit) $= 0.1 * 50 * 2 = 10$
$LOCKS\_ON\_SUBCLASS_E$ (Implicit) $= 0$
$LOCKS\_ON\_SUBCLASS_F$ (Implicit) $= 0.3 * 20 * 1 = 6$
$LOCKS\_ON\_SUBCLASS_G$ ((Implicit) $= 0.1 * 40 * 1 = 4$
$LOCKS\_ON\_SUBCLASS_H$ (Implicit) $= 0$
$LOCKS\_ON\_SUBCLASS_I$ (Implicit) $= 0$
$LOCKS\_ON\_SUBCLASS_J$ (Implicit) $= 0$
$LOCKS\_ON\_SUBCLASS_K$ (Implicit) $= 0$
$LOCKS\_ON\_SUBCLASS_L$ (Implicit) $= 0$
$LOCKS\_ON\_SUBCLASS_M$ (Implicit) $= 0$

In above example, LOCKS_ON_IMPLICIT and LOCKS_ ON_EXPLICIT are calculated as follows.

LOCKS_ON_IMPLICIT = LOCKS_ON_SUPERCLASS + LOCKS_ON_SUBCLASS (Implicit)
= 1,680 + 58= 1,738
LOCKS_ON_EXPLICIT = 292

Thus, since LOCKS_ON_IMPLICIT > LOCKS_ON_ EXPLICIT, it is better to use explicit locking for reducing locking overhead in this case.

4. Conclusions and Further Research Works

The typical OODBs can provide complex modeling power than traditional relational databases. Therefore, concurrency control schemes in OODBS are more complex than concurrency control schemes in traditional databases. Also, in typical OODBs, transactions are long-lived. It is more likely that an active transaction may block other concurrent transaction on the same database access. For this reason, in order to maintain or keep good database performance for OODBs, it is crucial to adopt a good concurrency control scheme that incurs less locking overhead.

In the previous works, two representative concurrency control schemes are adopted for dealing with class hierarchy, implicit locking and explicit locking, respectively. Both concurrency control schemes have different philosophy dealing with solving possible conflicts in class hierarchy. In this paper, we present the performance evaluation metrics for concurrency control schemes dealing with class hierarchy in OODBs. The proposed performance evaluation metrics are developed to determine which concurrency control scheme, implicit locking and explicit locking, can provide better performance for a given class hierarchy. We hope that our proposed performance evaluation metrics will be helpful to determine a right concurrency control scheme for a given class hierarchy in OODBs.

We have a plan on developing performance evaluation metrics dealing with composite object hierarchy. Usually a

composite object has more complicated relationships among shared and non-shared objects. It means that developing performance evaluation metrics could be more complicated. We will finally develop an integrated performance evaluation metrics for both of class hierarchy and class composition hierarchy.

References

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

[2] W. Jun and S. Hong, "Developing of a Concurrency Control Technique for Multiple Inheritance in Object-Oriented Databases", *Journal of Internet Computing and Services*, Vol. 15, No. 1, 2014. http://dx.doi.org/10.7472/jksii.2014.15.1.63

[3] P. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems", Reading, Massachusetts, Addison-Wesley, 1987.

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

[5] C. Malta and J. Martinez, "Automating Fine Concurrency Control in Object-Oriented Databases", *In Proc. of 9th IEEE Conf. on Data Engineering*, Vienna, Austria, pp. 23-260, Apr., 1993.

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

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

[8] W. Kim, E. Bertino and J. Garza, "Composite Object Revised", *ACM SIGMOD RECORD*, Vol. 18, No. 2, pp. 337-347, 1989.

◐ 저 자 소 개 ◑

전 우 천 (Woochun Jun)
1985년 서강대학교 전산학 학사
1987년 서강대학교 대학원 전산학 석사
1997년 미국 University of Oklahoma 전산학 박사
1998년~현재 서울교육대학교 컴퓨터교육과 교수
관심분야 : 정보영재, 장애인정보화교육, 정보통신윤리교육
E-mail: wocjun@snue.ac.kr

홍 석 기 (Suk-Ki Hong)
1985년 서강대학교 경제학과 학사
1996년 미국 University of Nebraska-Lincoln, Management 박사
2003년~현재 단국대학교 상경대학 교수
관심분야 : e-Business, e-Service, SCM, Current Engineering, Product Life Cycle Management
E-mail: skhong017@dankook.ac.kr