A Class Hierarchy Concurrency Control Technique in Object-Oriented Database Systems

Woochun Jun and Le Gruenwald
Dept. of Computer Science
Univ. of Oklahoma
Norman, OK 73072
gruenwal@cs.ou.edu

Abstract

In this paper, we present a locking-based concurrency control scheme for object-oriented databases (OODBs). Our scheme deals with class hierarchy which is an important property in OODBs. The existing concurrency controls for a class hierarchy perform well only for specific environments. Our scheme is based on so called special classes and can be used for any applications with less locking overhead than existing work.

1. Introduction

In an object-oriented databases (OODBs), there are two types of access to an object: *instance access* methods (instance read and instance write) and *class definition access* methods that provide class definition read and class definition write. [1,2]. Commutativity is a criterion widely used to determine whether a method can run concurrently with methods in progress on the same object. Two methods commute if their execution orders are not important (i.e., their execution orders do not affect their results). Two methods conflict with each other if they do not commute.

A concurrency control scheme allows multiusers rapid access to a database but incurs an overhead whenever it is invoked. This overhead may have a critical effect on OODBs where many transactions which consist of method invocations are long-lived. Thus, reducing the overhead is vital to improve transaction response time.

One of the major properties of an OODB is inheritance. That is, a subclass inherits definitions defined on its superclasses. Also, there is an *is-a* relationship between a subclass and its superclasses. Thus, an instance of a subclass is a specialization of its superclasses (and conversely, an instance of a superclass is a generalization of its subclasses) [5]. This inheritance relationship between classes forms a *class hierarchy*. While there are some operations on only one class such as class definition read or instance write on one instance, there are two types of operations on a class hierarchy: *class definition write and instance access to all or some instances of a given class and its subclasses* which we call IACH, meaning Instance Access to Class Hierarchy. A query is an example of IACH where a *query* is defined as instance read to a given class and its subclasses [5]. Due to inheritance, the definitions of the class' superclasses should not be modified, while a class and its instances are being accessed. Also, due to the *is-a* relationship between classes, the search space for a query against a class, C, may include the instances of all classes in the class hierarchy rooted at C as well as instances of C. Thus, for a locking based concurrency control scheme, when a class definition write or query is requested on some class, say C, it is necessary to get locks for all subclasses of C as well as C. We call MCA (Multiple Class Access) for class definition write and IACHs, and SCA (Single Class Access) for an operation to only one class such as class definition read and instance access to a single class.

In this paper, we present a locking-based concurrency control scheme for class hierarchy in OODBs. There are two approaches in the literature that deal with class hierarchy, *explicit locking* and *implicit locking*, both will be discussed in Section 2. These approaches may work well only for specific applications in OODBs. Explicit locking may have less locking overhead for transactions concerned only with SCA. On the other hand, implicit locking may have less locking overhead for transactions concerned only with MCA. Our scheme is based on a so called *special class*, which will be defined in Section 3, and which can be used for any applications with less locking overhead than both explicit locking and implicit locking.

2. Related Work

As discussed in Section 1, due to inheritance, class definition writes and IACHs on a class may need to access more than one class on a class hierarchy. There are two major existing approaches to

perform locking on a class hierarchy: *explicit locking* [2,10] and *implicit locking* [5,8,9]. In *explicit locking*, for an IACH involving a class, C, and all of its subclasses, and for a class definition write on a class C, a lock is set not only on the class C, but also on each subclass of C on the class hierarchy. For other types of access such as class definition read and instance access to a single class, a lock is set for only the class to be accessed (we call *target class*). Thus, for an MCA, transactions accessing a class near the leaf level of a class hierarchy will require fewer locks than transactions accessing a class near the root of a class hierarchy. Another advantage of *explicit locking* is that it can treat *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, in the same way. However, this technique increases the number of locks required by transactions accessing a class at a higher level in the class hierarchy.

In *implicit locking*, setting a lock on a class C requires extra locking on a path from C to its root as well as on C. *Intention* locks [3,7] are put on all the ancestors of a class before the target class is locked. An intention lock on a class indicates that some lock is held on a subclass of the class. For MCA on a target class, locks are not required for every subclass of a target class. It is sufficient to put a lock only on the target class (in single inheritance) or locks on the target class and subclasses of the target class which have more than one superclass (in multiple inheritance) [9]. Thus, it can reduce lock overhead over explicit locking. But, *implicit locking* requires a higher locking cost when a target class is near the leaf level in the class hierarchy due to intention lock overhead.

3. Proposed class hierarchy locking scheme

Our work is to develop a new class hierarchy locking scheme which can be used for any OODB applications with less locking overhead than both existing schemes, explicit locking and implicit locking. To achieve this, we designate some classes in the class hierarchy as *special classes*. We define a *special class* (SC) as a class on which class definition writes or IACHs are performed frequently. For our concurrency control purpose, how to determine if a class is a SC or not will be discussed in Section 4.

3.1. Lock Modes

We adopt instance level granularity for instance access and entire class object for class definition access, like Orion [5] and O_2 [2]. Below we show locks needed for different types of instance and class access. For convenience, we use lower-case letters and upper-case letters to name locks for an instance and for a class, respectively.

• instance read
- r lock for target instance
- (for SCA) TR lock means that some (not all) instances of a target class are *r* locked. An TR lock is set on a *target class* whenever a *r* lock is set on its instance.
- (for SCA) IR lock (on target class) means that all instances are *read* locked *implicitly*. Like both explicit locking and implicit locking, we reduce locking overhead by setting an IR lock on the target class, not individual instances, if the majority of instances are accessed.
- (for MCA) QR (*Query Read* on a target class) means that all instances of a target class and its subclasses are *read* locked as in implicit locking. We reduce locking overhead by setting an QR lock on only the target class, not setting IR lock on the all subclasses of the target class.
- (for MCA) PQR (Partial Query Read on a target class) means that some instances of a target class and its subclasses are read locked. For access to some instances of a target class and its subclasses, we put only PQR lock on a target class and each individual instances to be accessed are *r* locked.
- An intention lock ISR is set for every SC on the *superclass chain* from a target class to its root whenever IR or QR lock is set on the target class.
- An intention lock ISPR is set for every SC on the superclass chain from a target class to its root when TR or PQR lock is set on the target class.

• Instance write
- w lock for target instance
- (for SCA) TW lock (on target class) means that some (not all) instances of a target class are *w* locked. An TW lock is set on a *target class* whenever *w* lock is set on its instance.
- (for SCA) IW lock means that all instances of a target class are *w* locked *implicitly*.
- (for MCA) QW (Query Write on a target class) means that all instances of a target class and its subclasses are write locked.
- (for MCA) PQW (Partial Query Write on a target class) means that some instances of a target class and its subclasses are write locked. As in PQR lock, we set

only PQW lock on a target class and each individual instances to be accessed are *w* locked.
- ISW lock is set for every SC on the *superclass chain* from the target class to its root whenever an IW or QW lock is set on an instance or class.
- An intention lock ISPW is set for every SC on the superclass chain from a target class to its root when TW or PQW lock is set on the target class.

• Class definition write: CW (on target class),ISW (intention lock for each SC on the path from the target class to its root)

• Class definition read: CR (on target class), ISR (intention lock for each SC on the path from the target class to its root)

3.2. Commutativity Relation Table

In Tables 1 and 2, we provide commutativity relation among the lock modes introduced above. Y(Yes) and N (No) stand for *commute*, and *not commute*, respectively.

a) instance

		lock holder	
		r	*w*
lock requester	*r*	Y	N
	w	N	N

Table 1. Commutativity relation for locks on instances

b) Class

As in implicit locking, conflicts between MCAs, if at least one of the lock holder and requester requires locks only on class, conflict relationship is determined directly by read-read, read-write, write-write conflict. Otherwise (i.e., both require locks on class as well as instances), conflict is determined on individual instances. For conflicts between MCA and intention locks, conflicts are determined as if an intention lock were an actual real lock. For example, locking on CW and IMP-S-R on the same class will cause conflicts. Also, there is no conflict between SCA and an intention lock.

lock holder

		C	C	T	I	I	I	Q	P	T	I	I	I	Q
P														
		W	R	R	R	S	S	R	Q	W	W	S	S	W
Q														
								R	P		R		W	P
W														
									R					W
l	CW	N	N	N	N	N	N	N	N			N	N	N
o	CR			N	Y	Y	Y	Y			Y	Y	Y	
c	TR			N	Y	Y	Y	Y			Y	Y	Y	
k	IR			N	Y	Y	Y	Y			Y	Y	Y	N
	ISR	N	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
r	ISPR	N	Y	Y	Y	Y			Y	Y	Y	Y	Y	Y
e	QR	N	Y	Y	Y	Y			Y	Y	Y	N	N	N
q	PQR	N	Y	Y	Y	Y			Y	Y	Y	Y	N	N
q	TW	N	Y	Y	N	Y			Y	N	Y			Y
e	IW			N	Y	N	N	Y			Y	N	N	N
s	ISW	N	Y	Y	Y	Y			Y	N	N	Y	Y	Y
t	ISPW	N	Y	Y	Y	Y			Y	N	Y	Y	Y	Y
e	QW	N	Y	N	N	N			N	N	N			N
r	PQW	N	Y	Y	N	N			Y	N	Y			Y

Table 2. Commutativity table for locks on classes

3.3 Class hierarchy locking algorithm

Our locking-based concurrency control scheme is based on two-phase locking which requires each transaction to obtain a read (or write) lock on a data item before it reads (or writes) that data item, and not to obtain any more locks after it has released some lock [4]. For a given lock request on a class, say Y, we set locks on Y and all classes on the class hierarchy to which the class Y belongs as follows.

Step 1) locking on SCs
- For each *SC* (if any) through the superclass chain of Y, check conflicts and set an *intention lock* if

it commutes. If it does not commute, block the lock requester.

Step 2) Locking on a target class
If the lock request is SCA, check conflicts with locks set by other transactions and set one of T-R, T-W, IMP-R, IMP-W (depending on the lock request type) or CR (class definition read) on only the target class Y if it commutes and set an *r* or *w* lock on the instance to be accessed (which we call *target instance*) if *a* method is invoked on the instance and commute. If it does not commute, block the requester.

- If the lock request is an MCA, then, from class Y to the first *SC* (or leaf class if there is no SC) through the subclass chain of Y, check conflicts and set CW, QR, P-QR, QW or P-QW lock on each class if commute. If the class Y is a SC, then set a lock only on Y.
- If class Y has more than one subclass, perform the same step 2) for each subclass of Y.

3) Locks are released only if a transaction is committed or aborted.

For the correctness of our scheme, we can prove it by showing that, for any lock requester, any conflict with a lock holder is always detected [6].

4. Special Class Assignment

Assume that we have information on frequencies of access to each class in an OODB. For our scheme, we need to know only two types of access frequencies to each class: SCA and MCA. With those access frequencies for each class, we determine if the class is designated as a SC or not as follows.
Starting from each leaf class until all classes are checked.
step 1) If a class, say C, is a leaf, then do not designate it as a SC.
If a class' subclasses have been already checked (i.e., all of the subclasses have been determined for SC assignment), do the following:
for classes C and all of the subclasses, calculate the number of locks (N_1) when the class is designated as a SC
calculate the number of locks (N_2) when the class is not designated as a SC
step 2) Designate it as a SC only if $N_1 < N_2$. That is, the class can be a SC only if the number of locks can be reduced by doing so.

We can show that our scheme performs better than both explicit locking and implicit locking. That is, assuming that access frequencies are stable for each class, we show that our scheme incurs fewer or at least equal number of locks than both explicit locking and implicit locking. We omit the formal proof due to lack of space.

5. Further work

In our work, locking granularity for instance access is instance object and for class definition access is class object. We are currently developing a class hierarchy locking scheme with finer granularity. That is, we are adopting attribute level granularity instead of instance, and finer class definition instead of entire class object, in order to provide better concurrency among transactions.

References

[1] D. Agrawal and A. E. Abbadi, "A Non-restrictive Concurrency Control for Object-Oriented Databases", 3rd Int. Conf. on Extending Data Base Technology, Vienna, Austria, Mar. 1992, pp 469 - 482.

[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, Mar. 1990, pp. 363 - 377.

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

[4]. K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The notion of consistency and predicate locks in a database system", Communication of ACM, Vol. 19, No. 11, Nov. 1976, pp. 624 - 633.

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

[6]. W. Jun and L. Gruenwald, "A Flexible Class Hierarchy Locking Technique in Object-Oriented Database System", 11th Int. Conf. on Computers and Their Applications, San Francisco, Mar. 1996, pp. 191 - 196.

[7]. H. F. Korth and A. Silberschartz, A. *Database System Concepts*, 2nd Edition, McGraw Hill, 1991.

[8]. 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, Feb. 1996, pp. 144 - 156.

[9]. C. Malta and J. Martinez, "Controlling Concurrent Accesses in an Object-Oriented Environment", 2nd

Int. Symp. on Database Systems for Advanced Applications, Tokyo, Japan, Apr. 1991, pp. 192 - 200.
[10]. C. Malta and J. Martinez, "Automating Fine Concurrency Control in Object-Oriented Databases", 9th IEEE Conf. on Data Engineering, Vienna, Austria, Apr. 1993, pp. 253- 260.
[11]. M. Ozsu and P. Valduriez, Principles of Distributed Database Systems, Prentice Hall, 1991.