社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－DPS－117 (7)
2004－CSEC－ 24 (7)
2004／3／4

Moderate Concurrency Control in Distributed Object Systems

Yousuke Sugiyama, Tomoya Enokido, and Makoto Takizawa
Dept. of Computers and Systems Engineering
Tokyo Denki University, Japan
{sugi, eno, taki}@takilab.k.dendai.ac.jp

Abstract

In object-based systems, objects are distributed in multiple object servers like database servers interconnected with communication networks. Objects are manipulated only through their own methods. Methods are procedures for manipulating an object. We first extend traditional lock modes for read and write on simple objects like files and tables to methods on objects. We introduce new types of conflicting relations among methods, availability and exclusion ones. Then, we define a partially ordered relation on lock modes showing which one is weaker than another for a pair of modes. We discuss a moderate concurrency control algorithm for concurrently manipulating distributed objects. Before manipulating an object through a method, the object is locked in a weaker mode than a an intrinsic mode of the method. Then, the lock mode is escalated to the method mode. The weaker the initial mode is the more concurrency is obtained but the higher frequently deadlock occurs.

分散オブジェクト上における中庸な同時実行制御プロトコル

杉山 洋介　榎戸 智也　滝沢 誠
東京電機大学大学院理工学研究科情報システム工学専攻
E-mail {sugi, eno, taki}@takilab.k.dendai.ac.jp

情報システムでは多種多様なオブジェクトサーバが高速通信ネットワークにより相互接続されている。オブジェクトはデータと手続きがカプセル化されたものである。本論文では、ファイルやテーブルのような単純なオブジェクトに対して議論されてきたロックモードを、複雑なメソッドに拡張する。ついで、可用性と排他性によるメソッド間の新しい競合関係について提案する。これを基に、ロックモードの強弱についての半順序関係を定義し、中庸な同時実行制御アルゴリズムについて述べる。中庸な同時実行制御においては、オブジェクトに操作を行う前に弱いロックをかけ、操作終了後にメソッド固有のロックモードに強化される。

1 Introduction

Object-based systems [6] are composed of objects which are distributed in networks. Each object is an encapsulation of data and methods for manipulating the data. A method is a procedure which is more complex than *read* and *write*. An object is allowed to be manipulated only through methods supported by the object. A method op_1 *conflicts* with another method op_2 on an object o if and only if (iff) the result obtained by performing the methods op_1 and op_2 depends on the computation order of op_1 and op_2 on the object o. A conflicting relation among methods is specified based on the semantics of an object in defining the object.

Lock modes based on conflicting relations are discussed by Korth [4]. If a pair of methods conflict with one another on an object, both the methods cannot be concurrently performed, i.e. the lock modes of the methods are *exclusive*. Otherwise, the modes are *compatible*. A conflicting relation among methods on an object may not be symmetric although the conflicting relation among primitive read and write methods are symmetric. We introduce novel types of relations, *exclusion* and *availability* relations among methods to lock an object based on the conflicting relation. The exclusion relation shows how exclusively each method can be performed. The availability relation indicates how frequently an object can be locked to perform each method. In traditional *pessimistic* approaches [2, 4], every object is locked before manipulated by a transaction. An object is locked in *write* (w) and *read* (r) modes before the object is written and read, respectively. In another *optimistic* approach [5], a transaction manipulates an object without locking the object. When the transaction commits, every object manipulated is validated that the object has not been manipulated in a conflicting way by other transactions. If validated, the transaction commits. Otherwise the transaction aborts.

An object is first read and then written by a transaction in order to update the object in typical transactions. In order to increase the throughput in the pessimistic approach, every object is first locked in a *read* (r) mode even if the object is manipulated by a *write* (w) method. Then,

a transaction manipulates the object. When a transaction commits, lock modes of objects manipulated by the transaction are changed up to a write mode. If successfully changed, the transaction commits. Otherwise, the transaction aborts. This way can be said to be *moderate* in between the pessimistic and optimistic ways.

In this paper, we extend the optimistic locking protocol for simple *read* and *write* methods to objects which support more abstract levels of methods. We discuss a novel type of moderate concurrency control algorithm for distributed objects. In the algorithm, an object in an object server is first locked in a *weaker* mode than a method before a transaction manipulates the object in the method. Then, the object is locked in an intrinsic mode of the method on commitment of the transaction. We discuss what a *stronger* mode means on the basis of availability and exclusion relations of lock modes.

In section 2, we present a system model and discuss conflicting relations on methods. In section 3, we discuss lock modes. In section 4, we discuss a moderate concurrency control protocol. In section 5. we discuss how to evaluate the protocol.

2 System Model

A system is composed of clients and object servers D_1, $\cdots$, D_n ($n \geq 1$) which are interconnected in reliable communication networks [Figure 1]. Each object server D_i supports applications with a collection of objects [6]. A database server [7] is a typical example of object server. An object is an encapsulation of data and methods. A method is a procedure for manipulating data in an object. An object is allowed to be manipulated only through its own methods. A pair of methods op_1 and op_2 *conflict* with one another on an object o iff the result obtained by performing the methods op_1 and op_2 depend on the computation order of op_1 and op_2. If some pair of methods are concurrently performed on an object, the object gets inconsistent or inconsistent results of the method are obtained. Here, the methods cannot be concurrently performed, i.e. exclusively performed on the object o. Thus, there are two aspects on conflicting relations to realize the serializability [2] and multiple exclusion, respectively.

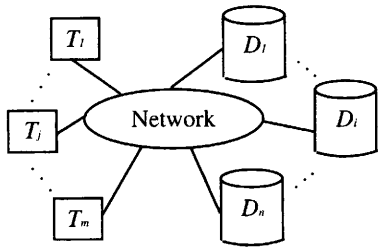

Figure 1. Distributed servers.

Application programs are performed on clients and application servers in 2-tier and 3-tier client server models, respectively. An application program issues a method *op* to manipulate an object o in an object server D_i. The method *op* is performed on the object o. Then, the response with the result obtained by performing the method *op* is sent back to the application program.

A *transaction* shows execution state of an application program which issues method requests to object servers and receives responses form the objects server. A transaction is modeled to be an atomic sequence of methods issued to objects, which satisfies ACID (atomicity, consistency, isolation, durability) properties [1]. A transaction T_1 *precedes* another transaction T_2 ($T_1 \rightarrow T_2$) iff T_1 are T_2 issue conflicting methods op_1 and op_2 to an object o, respectively and op_1 is performed before op_2. A collection $\mathbf{T}$ of transactions $T_1, \cdots, T_n$ are *serializable* iff both $T_i \rightarrow T_j$ and $T_j \rightarrow T_i$ do not held for every pair of transactions T_i and T_j in the transaction set $\mathbf{T}$ [1, 2]. That is the precedence relation is asymmetric. In order to make objects consistent, multiple transactions are requested to be serializable.

3 Lock Modes

3.1 Availability and exclusion sets

In traditional concurrency control theories [2], a conflicting relation among *read* and *write* methods on a simple object like *file* and *table* is symmetric. For example, *read* conflicts with *write* while *write* conflicts with *read*. Korth [4] discusses an asymmetric conflicting relation with a new mode u to update an object in addition to $read(r)$ and $write(w)$ modes. Here, w and u conflict with one another. However, the mode u is compatible with r but r conflicts with u so that a *read* mode can be escalated to a *write* mode without deadlock in a transaction. A transaction first locks an object in an u mode and reads the object. Then the transaction escalates the lock mode to a w mode to write the object. Even if other transactions lock the object in an r mode, the transaction can lock the object in a u mode. However, no transaction can neither lock the object in an r mode nor w mode after the transaction holds the lock in the u mode. Transactions can update objects without deadlock.

Suppose a method *op* is issued to an object o. Here, the object o is locked in a mode. Let $\mu(op)$ denote a lock mode if a method *op*. If the object o is successfully locked in the mode $\mu(op)$, the method *op* is performed on the object o. Otherwise, the method *op* is kept blocked. Now, we define a conflicting relation among lock modes on an object o. Let $\mathbf{M}_o$ be a set of lock modes on an object o.

[Definition] A lock mode $\mu(op_1)$ *conflicts with* a lock mode $\mu(op_2)$ on an object o ($\mu(op_1) \rhd \mu(op_2)$) iff op_1 cannot be performed while op_2 is being performed on the object o. □

The conflicting relation $\triangleright$ on the mode set $\mathbf{M}_o$ ($\triangleright \subseteq \mathbf{M}_o^2$) is neither symmetric nor transitive. A mode $\mu(op_1)$ may not conflict with a method $\mu(op_2)$ ($\mu(op_1) \not\triangleright \mu(op_2)$) even if $\mu(op_2) \triangleright \mu(op_1)$. A lock mode $\mu(op_1)$ is *compatible* with a lock mode $\mu(op_2)$ ($\mu(op_1) \square \mu(op_2)$) if $\mu(op_1)$ does not conflict with $\mu(op_2)$, i.e. $\mu(op_1) \not\triangleright \mu(op_2)$. The compatibility relation $\square$ may not be symmetric either. If a conflicting relation $\triangleright$ is symmetric on an object o, a relation $\diamondsuit$ shows a symmetric conflicting relation on the mode set $\mathbf{M}_o$.

Let $\mathbf{P}_o$ be a set of methods supported by an object o. We define an *availability* set $A(op)$ ($\subseteq \mathbf{P}_o$) and *exclusion* set $E(op)$ ($\subseteq \mathbf{P}_o$) of methods for a every method op on an object o as follows:

- $A(op) = \{op_1 \mid \mu(op)$ conflicts with $\mu(op_1)$ ($\mu(op) \triangleright \mu(op_1)$)$\}$.
- $E(op) = \{op_1 \mid \mu(op_1)$ conflicts with $\mu(op)$ ($\mu(op_1) \triangleright \mu(op)$)$\}$.

If the conflicting relation $\triangleright$ is not symmetric, the availability set $A(op)$ is different from the exclusion set $E(op)$. If the *conflicting* relation $\triangleright$ is symmetric, $A(op) = E(op)$. Here, let a conflicting set $C(op)$ be $A(op)$ (= $E(op)$), i.e. $C(op) = \{op_1 \mid \mu(op_1) \diamondsuit \mu(op)\}$. For every pair of methods op_1 and op_2, $op_1 \in C(op_2)$ iff $op_2 \in C(op_1)$.

First, a method op is issued to an object o. If any method in the availability set $A(op)$ is being performed on an object o, a method op cannot be performed, i.e. blocks until no method in $A(op)$ is performed. The larger the availability set $A(op)$ is, the less frequently a method op can be performed. Suppose $A(op_1) \subseteq A(op_2)$ for a pair of methods op_1 and op_2 on an object o. Here, op_2 cannot be performed if op_1 cannot be performed, i.e. some method conflicting with op_1 is being performed on an object o. However, even if op_2 cannot be performed, op_1 may be performed. Here, op_1 is referred to as *more available* than op_2 ($op_1 \succ_A op_2$). op_1 is equivalent (*A-equivalent*) with op_2 with respect to the available set A ($op_1 \equiv_A op_2$) iff $A(op_1) = A(op_2)$. $op_1 \succeq_A op_2$ iff $op_1 \succ_A op_2$ or $op_1 \equiv_A op_2$.

Next, suppose a method op is now being performed on an object. Even if a method op_1 in the exclusion set $E(op)$ is issued to the object o, op_1 cannot be performed. The larger the exclusion set $E(op)$ is, the more exclusively a method op is performed, i.e. the fewer number of methods can be concurrently performed with op. Suppose $E(op_1) \subseteq E(op_2)$ for a pair of methods op_1 and op_2 on an object o. Here, if some method issued cannot be performed since the method op_1 is now being performed, the method cannot be performed if op_2 is being performed. Here, op_2 is referred to as *more exclusive* than op_1 ($op_2 \succ_E op_1$). op_1 is equivalent (*E-equivalent*) with op_2 with respect to the exclusion set E ($op_1 \equiv_E op_2$) iff $E(op_1) = E(op_2)$. $op_1 \succeq_E op_2$ iff $op_1 \succ_E op_2$ or $op_1 \equiv_E op_2$. op_1 is *equivalent* with op_2 ($op_1 \equiv op_2$) iff $op_1 \equiv_A op_2$ and $op_1 \equiv_E op_2$.

[Definition] Let op_1 and op_2 be methods on an object o.

1. A mode $\mu(op_1)$ is *more available* than $\mu(op_2)$ ($\mu(op_1) \succ_A \mu(op_2)$ iff $op_1 \succ_A op_2$.
2. $\mu(op_1)$ is *more exclusive* than $\mu(op_2)$ ($\mu(op_1) \succ_E \mu(op_2)$) iff $op_1 \succ_E op_2$. $\square$
3. $\mu(op_1)$ is *stronger* than $\mu(op_2)$ ($\mu(op_1) \succ_E \mu(op_2)$) iff $op_1 \succ_A$ and $op_1 \succ_E op_2$. $\square$

Methods supported by an object o are partially ordered in the availability $\succ_A$ and exclusion $\succ_E$ relations. Let $\mathbf{P}_o$ be a set of methods supported by an object o. For every pair of methods op_1 and op_2 in $\mathbf{P}_o$, $op_1 \cup_A op_2$ and $op_1 \cup_E op_2$ denote least upper bounds (*lubs*) of op_1 and op_2 with respect to the availability and exclusion relations $\succ_A$ and $\succ_E$, respectively. $op_1 \cap_A op_2$ and $op_1 \cap_E op_2$ show greatest lower bounds (*glbs*) of op_1 and op_2 with respect to the availability and exclusion relations $\succ_A$ and $\succ_E$, respectively.

Let us consider an example of r, w, and u modes [4]. The availability and exclusion sets for the methods are given in Table 1. Here, since $A(u) \subseteq A(r) \subseteq A(w)$, r is more available than w and u is more available than r and w ($r \succ_A u \succ_A w$). Since $E(r) \subseteq E(u) \subseteq E(w)$, w is more exclusive than r and u while u is more exclusive than r ($w \succ_E u \succ_E r$). $r \cup_A u = r \cap_E u = u$ and $r \cap_A u = r \cap_E u = r$.

Table 1. Availability and exclusion sets

	A	E
r	$\{w, u\}$	$\{w\}$
w	$\{w, r, u\}$	$\{w, r, u\}$
u	$\{w\}$	$\{w, r\}$

3.2 Conflicting set

Let us consider case the conflicting relation $\triangleright$ is symmetric, i.e. conflicting relation $\diamondsuit$ holds and $A(op) = E(op)$ for every method op on an object o. Here, the conflicting set $C(op)$ is defined to be $A(op)$ ($= E(op)$) for every method op on an object o.

[Definition] A method op_1 is *stronger* than another method op_2 ($op_1 \succ op_2$) iff $C(op_1) \subset C(op_2)$ $\square$.

In another word, op_2 is *weaker* than op_1. op_1 and op_2 are *equivalent* ($op_1 \equiv op_2$) iff $C(op_1) = C(op_2)$. $op_1 \succeq op_2$ iff $op_1 \succ op_2$ or $op_1 \equiv op_2$.

Let $\cup$ and $\cap$ show *least upper bound* (lub) and *greatest lower bound* (glb) of methods with respect to the strength relation $\succeq$, respectively. In the r and w modes, $C(r) = \{w\}$ and $C(w) = \{r, w\}$. Since $C(r) \subseteq C(w)$, a *write* (w) mode is stronger than a *read* (r) mode ($w \succeq r$). $r \cup w$ and $r \cap w$ show modes w and r, respectively.

Next, let us consider a *counter* object c which supports methods *initialize* (*ini*), *increment* (*inc*), *decrement* (*dec*), and *check* (*chk*). The conflicting relation $\Diamond$ is symmetric. For example, a pair of methods *chk* and *inc* conflict with one another, $chk \Diamond inc$. A pair of methods *inc* and *dec* are compatible withe one another, $inc \not\Diamond dec$. Table 2 shows the conflicting sets C for methods. For example, $C(ini) = \{ini, inc, dec, chk\}$ and $C(inc) = \{ini, chk\}$. Here, *ini* is stronger than *inc* ($ini \succeq inc$) since $C(inc) \subset C(ini)$. *inc* and *dec* are equivalent ($inc \equiv dec$) since $C(inc) = C(dec)$. $chk \cup inc$ is ini. $chk \cap inc$ show a virtual mode $\{ini\}$.

A conflicting mode set $CM(\mu)$ is defined to be a set of modes conflicting with a mode μ in $\mathbf{M}_o$. For example, $CM(ini) = \{ini, chk, inc, dec\}$ and $CM(inc) = CM(dec) = \{ini, chk\}$. A lock mode $\mu(op)$ is also represented by the conflicting set $CM(\mu op)$. For example, the lock mode of the methods *inc* and *dec* can be written in a set $\{ini, chk\}$.

Table 2. Conflicting sets

	C
ini	$\{ini, chk, inc, dec\}$
chk	$\{ini, inc, dec\}$
inc	$\{ini, chk\}$
dec	$\{ini, chk\}$

4 Moderate Locking Protocol

4.1 Traditional approaches

We overview traditional pessimistic [2] and optimistic [3,5] concurrency control protocols on distributed objects in order to extend the traditional protocols [2,3,5] to objects with methods in an object-based system. In the optimistic approach [5], objects are manipulated by transactions as follows:

1. A transaction T manipulates, i.e. reads and writes an object without locking.
2. On completion of the the transaction T, it is validated whether or not the object is manipulated by other conflicting transactions during the execution of T. If validated, the transaction T commits. Otherwise, T aborts.

Jing et. al. [3] discuss another optimistic protocol for mobile databases.

1. If a transaction T issues a *read* request to an object o in an object server D_i, the object o is locked in a *read* (r) mode. If succeeded, *read* is performed on the object o.
2. If a transaction T issues a *write* request on an object o, the object o is locked in an r mode. The *write* method is performed on the object o if succeeded.
3. If a transaction T commits, the r locks on objects written by the transaction T are changed with *write* (w) mode. If changed, the transaction T commits. Otherwise, the transaction T aborts.

In the pessimistic way, a transaction T locks an object before reading and writing the object. If *read* and *write* are issued to an object, the object is locked in read (r) and write (w) modes, respectively. A transaction issues no lock after releasing a lock, i.e. on completion of the transaction, all the locks are released in the strict two-phase locking protocol [2].

In the most optimistic way [5], objects are manipulated without locking. In the most pessimistic way, objects are written after locked in a w mode. In another way, an object is first locked in a weaker mode, i.e. r mode. Then, the lock mode is escalated to the w mode. This way is referred to as *moderate*, which is more optimistic than the pessimistic way and more pessimistic than the optimistic way. We extend the moderate protocol for *read* and *write* to abstract methods on distributed objects.

4.2 Locking protocol

We discuss an algorithm of a moderate concurrency control protocol on distributed objects. We extend the traditional concurrency control algorithm for primary methods *read* and *write* to methods, i.e. procedures on objects. We consider an object o which supports a collection $\mathbf{P}_o$ of methods. Let $\mathbf{M}_o$ be a set of lock modes $\{\mu(op) \mid op \in \mathbf{P}_o\}$. First, we assume all the conflicting relations of methods are symmetric on every object. First, let $\perp$ be a *bottom* mode, i.e. $\perp = \cup \{ \mu \mid \mu \in \mathbf{M}_o \}$. Let $\top$ denote a *top* mode $\top = \cap \{\mu \mid \mu \in \mathbf{M}_o \}$. $\top$ shows the most exclusive lock which no other method be performed on an object if the object is locked in the mode $\top$. On the other hand, $\perp$ indicates that every other method can be performed with the mode $\perp$. Here, $C(\perp) = \phi$ and $C(\top) = \mathbf{M}_o$. If an object supports only *read* and *write*, $\top$ is a w mode and $\perp$ is an r mode.

Suppose a transaction T issues a method op_i to the object o. The object o is locked and manipulated in a method op by a transaction T as follows:

[Locking protocol]

1. Let β be some lock mode such that $\perp \preceq \beta \preceq \mu(op)$. The object o is locked in the mode β.
2. If locked, the method op is performed on the object o. Otherwise, the transaction T waits.
3. When the transaction T commits, the lock mode of the object o is escalated to the mode $\mu(op)$. If succeeded, the transaction T commits. Otherwise T aborts. □

If $\beta = \perp$, the protocol is the most optimistic. If $\beta = \mu(op)$, the protocol is the most pessimistic. We can take another mode β such that $\perp \preceq \beta \preceq \mu(op)$ in the moderate approach. The stronger the mode β is, the more pessimistic the protocol is.

A conflicting mode set $CM(\mu(op))$ for a method op is defined to be a set $\{\mu(op') \mid op'$ conflicts with op on an object o $(op' \diamondsuit op)\}$ $(\subseteq \mathbf{M}_o)$ of lock modes. $CM(\mu)$ is a set of lock modes conflicting with a mode $\mu \in \mathbf{M}_o$. This means that an object o cannot be locked in any mode $\mu(op')$ is $CM(\mu)$ if the object o is locked in a mode μ.

Let $NM(\mu)$ be a set of lock modes which are compatible with a lock mode μ. Here, $CM(\mu) \cap NM(\mu) = \phi$ and $CM(\mu) \cup NM(\mu) = \mathbf{M}_o$. In the protocol, $CM(\beta)$ should be a subset of $CM(\mu(op))$, i.e. $CM(\beta) \subseteq CM(\mu(op))$. Each lock mode μ can be represented in a set $CM(\mu)$. A pair of modes μ_1 and μ_2 are equivalent $(\mu_1 \equiv \mu_2)$ iff $CM(\mu_1) = CM(\mu_2)$. We have to discuss which subset of $CM(\mu(op))$ to be selected as an initial lock mode β.

We introduce a concept of the *weight* $\mid \mu \mid$ of a lock mode μ. The weight $\mid \mu \mid$ shows the usage ratio, i.e. probability that a lock mode of a method issued is μ. The more frequently methods of the mode μ are issued, the larger $\mid \mu \mid$ is. For a set $\mathbf{M}$ of lock modes, $\mid \mathbf{M} \mid$ is defined to be $\Sigma_{\mu \in \mathbf{M}} \mid \mu \mid$. The weight $\mid \mu \mid$ of a lock mode μ is normalized as $\mid \mu_1 \mid + \cdots + \mid \mu_h \mid = 1$ where $\mathbf{M}_o = \{ \mu_1, \cdots, \mu_h \}$, i.e. $\mid \mathbf{M}_o \mid = 1$.

Now, suppose that an object o is locked in a mode $\mu_1 \in \mathbf{M}_o$ and then the lock mode is escalated to another mode $\mu_2 \in \mathbf{M}_o$ where $\mu_2 \succeq \mu_1$. The more number of lock modes are compatible with the mode μ_1, the more number of transactions can manipulate the object o. However, if transactions lock the object o in lock modes which are compatible with the mode μ_1 but conflict with μ_2, the lock mode μ_1 cannot be escalated to μ_2. Here, deadlocks may occur. $\mid NM(\mu_1) \mid$ shows probability that an object o can be locked by a method issued after an object o is locked in a mode μ_1. $\mid NM(\mu_1) \cap CM(\mu_2) \mid$ indicates probability that a lock mode held after an object o is locked in a mode μ_1 conflicts with μ_2, i.e. the lock mode μ_1 on the object o cannot be escalated to μ_2. For example, suppose $\mid ini \mid = 0.01$, $\mid chk \mid = 0.59$, $\mid inc \mid = 0.3$, and $\mid dec \mid = 0.2$. $\mid CM(dec) \mid = \mid \{ini, chk\} \mid = \mid ini \mid + \mid chk \mid = 0.6$ and $\mid NM(dec) \mid = 0.4$.

The following lock mode β in the power set $2^{CM(\mu(op))}$ is taken:

- $\mid NM(\beta) \mid$ is the maximum and $\mid NM(\beta) \cap CM(\mu(op)) \mid \leq \tau$ where the maximum allowable conflicting ratio τ.

The maximum allowable conflicting ratio τ is given by the designer of the system.

A transaction may manipulate an object o through multiple methods. For example, a transaction *increments* a counter object after *checking* the object. Here, suppose that a transaction T holds an object o in a mode μ_1 and then issues a lock request μ_2 to manipulate the object o. The lock mode of the object is changed as follows:

1. If $\mu_1 \succeq \mu_2$ or $\mu_2 \succeq \mu_1$, the lock mode μ_1 on the object o is not changed.
2. Otherwise, the lock mode μ_1 is escalated to the least upper bound $\mu_1 \cup \mu_2$.

The lock mode on an object is monotonically escalated but never descended in a transaction. [4].

5 Evaluation

We evaluate the moderate concurrency control protocol in terms of throughput and number of transactions to be aborted due to deadlocks. We present a way for evaluation the protocol. A system includes m objects $o_1, \cdots, o_m$ $(m \geq 1)$ and each object o_i supports k_i $(k_i \geq 1)$ types $op_{i1}, \cdots, op_{ik_i}$ of methods. Here, let $\mathbf{P}$ be a set of all methods $\mathbf{P}_{o_1} \cup \cdots \cup \mathbf{P}_{o_m}$ in the system. We assume that a conflicting relation C_i among methods is symmetric on every object o_i.

First, the conflicting relation C_i is randomly created. Here, the conflicting ratio σ_i is defined to be $\mid C_i \mid / k_i^2$. It is randomly decided whether not op_s and op_t conflict with one another for every pair of methods op_s and op_t on the object o given the conflicting ratio σ_i.

In the evaluation, we assume every object supports the same number of methods, i.e. $k_1 = \cdots = k_m = k$ and the same conflicting ratio $\sigma_1 = \cdots = \sigma_m = \sigma$.

Multiple transactions $T_1, \cdots, T_h$ manipulate objects o_1, $\cdots$, o_m. Each transaction T_s serially invokes l_s methods on the objects. A transaction T_s randomly selects a method on an object every τ_s time units. T_s is a sequence of methods $op^{s1}, \cdots, op^{sl_s}$, where each $op^{sj} \in \mathbf{P}$. Here, we assume every transaction issues the same number of methods, i.e. $l_1 = \cdots = l_h = l$ every same τ time units, i.e. $\tau_1 = \cdots = \tau_h = \tau$.

Suppose a transaction T_s issues a method op_{ij} to an object o_i. Here, the object o_i is first locked in a mode $\beta_{ij} (\preceq \mu(op_{ij}))$. If succeeded, T_s issues another method. If not, the transaction T_s waits. Finally, the transaction T_s commits. Here, a lock for a method op_{ij} is changed to the mode $\mu(op_{ij})$. If succeeded, all the locks held by the transaction T_s are released. Otherwise, T_s waits. If transactions are deadlocked, a deadlocked transaction is aborted.

In this simulation, all the objects are stored in one object server. The transactions $T_1, \cdots, T_h$ issue methods to the object server. We measure the throughput [methods/time unit] and the number of transactions aborted due to deadlock and the total throughput. We are now making simulation.

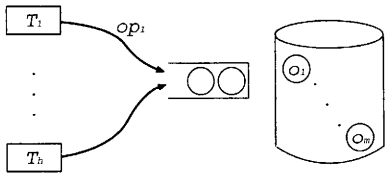

Figure 2. Evaluation

6 Concluding Remarks

We discussed the moderate concurrency control for distributed objects where objects are manipulated by methods. The moderate approach takes a position in between the pessimistic like two-phase locking protocol one and the optimistic one. We defined conflicting and exclusive relations among lock modes. We discussed the moderate concurrency control protocol by extending the traditional pessimistic and optimistic protocols for read and write methods to objects with methods.

References

[1] P. A. Bernstain, V. Hadzilaces, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[2] J. Gray. Notes on Database Operating Systems. *Lecture Notes in Computer Science*, (60):393–481, 1979.

[3] J. Jing, O. Bukhres, and A. Elmagarmid. Distributed Lock Management for Mobile Transacitons. *Proc. of the 15th IEEE International Conf. on Distributed Computing Systems (ICDCS-15)*, pages 118–125, 1995.

[4] H. F. Korth. Locking Primitives in a Database System. *Journal of the Association for Computing Machinery*, 30(1):55–79, 1983.

[5] H. T. Kung and J. T. Robinson. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems*, 6(2):213–226, 1981.

[6] J. Niwbray and R. Zahari. *The Essential CORBA*. Wiley, 1995.

[7] Oracle8i Concepts Vol. 1. 1999. Release 8.1.5.

[8] K. Tanaka and M. Takizawa. Quorum-based Locking Protocol for Replicas in Object-based Systems. *Proc. of the 5th IEEE International Symp. on Autonomous Decentralized Systems (ISADS' 2001)*, pages 196–203, 2001.