Optimistic Locking in Replicated Objects

Kyouji Hasegawa, and Makoto Takizawa
Tokyo Denki University
E-mail {kyo, taki}@takilab.k.dendai.ac.jp

The system is composed of multiple objects. Each object supports more abstract operations than the low-level read and write operations. The objects are replicated to increase the performance, reliability, and availability. In this paper, we discuss a synchronization method to make multiple replicas of objects mutually consistent. In the traditional optimistic two-phase locking (O2PL), every replica is first locked in a read mode to write an object. Finally, all the replicas are locked in a write mode when the transaction commits. We propose a novel optimistic object-based locking (OOBL) method based on the abstract operations. Here, the number of the replicas locked is decided based on the access frequency and the strength of the lock mode.

多重化されたオブジェクトに対する楽観的ロック

長谷川 享二　滝沢 誠
東京電機大学理工学部経営工学科
E-mail {kyo, taki}@takilab.k.dendai.ac.jp

現在の情報システムは、複数の計算機が通信網で相互接続された分散型の形態となっている。分散システム内の資源はオブジェクトとしてモデル化される。オブジェクトは、信頼性、可用性、性能を向上させるために、複数の計算機へ多重化される場合がある。オブジェクトの他の端末内へのコピーはレプリカと呼ばれ、レプリカ間の一致性を保つ必要がある。一致性を保つ方式としては、ロック方式が提案されている。しかしロック方式では、演算の競合が稀であるような応用に対しては、性能の低下が問題となる。これに対し演算間の競合が稀であると仮定して、レプリカ間の一致性を保つ方式が楽観的なロック方式である。本論文では、楽観的ロック手法を用い、ロック方式の問題点である性能の低下を防ぎ、トランザクションのアボート確率を減少させる方法を提案する。

1 Introduction

The distributed applications are realized by the cooperation of multiple objects $o_1, \ldots, o_n$ which exchange messages through the communication network. Each object o_i supports abstract operations for manipulating the state of o_i. The objects have to be mutually consistent in the presence of multiple accesses to the objects.

Many kinds of concurrency control methods [2] are discussed so far. In the famous two-phase locking (2PL) protocol [2,3], the transactions lock the objects before computing the operations on the objects. Most systems adopt the strict 2PL protocol where the locks obtained are released at the end of the transactions in order to resolve the cascading abort. Otherwise, the cascading abort of transactions may occur. One of difficulties in the distributed applications like the groupware [4] is that the objects are locked for a longer time. The optimistic concurrency control [8] is discussed to reduce the overhead implied by the locking. Here, every transaction T manipulates objects without locking the objects. When the transaction T ends up, T commits unless every object manipulated by T is manipulated by other transactions in the modes conflicting with T. Here, only *read* and *write* operations are considered.

In order to increase the reliability, availability, and performance of the system, the objects in the system are replicated. In the mobile database systems [1], the objects are also replicated by cashing the data in the fixed servers into the mobile clients in order to resolve the disconnected operations [5,7]. Here, it is critical to make the replicas of the object mutually consistent. Jing [6] discusses an optimistic two-phase locking (O2PL) method to maintain the mutual consistency among the replicas. In the O2PL, if a transaction T would write an object o, T locks all the replicas of o in a read (*Rlock*) mode. If locked, T manipulates the replicas. When T commits, T tries to convert the *Rlock* mode on the replicas to a write (*Wlock*) mode. If succeeded, T commits. Otherwise, T aborts.

The distributed applications are currently modeled in an object-based concept. That is, the applications are modeled to be collections of objects cooperating with each other. CORBA [10] is becoming a standard framework of the distributed applications. The objects can support more abstract operations than traditional low-level *read* and *write* operations. For example, a *Bank* object supports *Deposit* and *Withdraw* operations which are realized by *read* and *write* operation on the internal files. The objects can be manipulated only through the operations supported by the objects. In this paper, before each object is manipulated by an operation *op*, the object is locked in an abstract mode corresponding to *op*, e.g. *Deposit* mode for the *Deposit* operation of the *Bank* object. The conflicting relation between the lock modes is defined based on the conflicting operations. That is, if an operation op_1 conflicts with op_2 in an object o, the lock mode of op_1 conflicts with op_2. We discuss a novel optimistic concurrency control to maintain the mutual consistency among the replicas. In this paper, we propose a novel optimistic locking scheme for the replicated objects, named *OOBL* (*optimistic object-based locking*) protocol. The number of replicas to be locked depends on how *strong* the lock mode of the operation is and how *frequently* the operation is invoked. The stronger and more frequently used the lock modes are, the fewer replicas are locked.

In section II, we present the system model. In section III, we discuss the *OOBL* protocol.

2 System Model

2.1 Objects

A distributed system is composed of multiple objects $o_1, \ldots, o_n$ which are cooperating by exchanging messages in the communication network N. Let O be a set of objects in the system, i.e. $O = \{o_1, \ldots, o_n\}$ $(n \geq 1)$. The communication network N supports every pair of objects o_i and o_j with a reliable, bidirectional channel $\langle o_i, o_j \rangle$. Thus, o_i can exchange messages with o_j by the channel $\langle o_i, o_j \rangle$ without any message loss in the sending order.

A transaction T sends a request op_i to an object o_i. On receipt of op_i, o_i computes op_i. Here, op_i may invoke an operation op_{ij} on another object o_{ij}. o_i sends the response of op_i back to T if the computation of op_i completes. T is an atomic sequence of operations. T *commits* only if all the operations invoked by T successfully complete, i.e. operations commit. The operation op invoked by T commits only if all the operations invoked by op commit. op is also an atomic unit of computation. Thus, the operations are *nested*, i.e. nested transaction [9].

Each object o_i supports a set τ_i of operations $op_{i1}, \ldots, op_{il_i}$ for manipulating o_i. o_i is encapsulated so that o_i can be manipulated only through the operations supported by o_i. Let $op(s)$ denote a state obtained by applying op to a state s of o_i. An operation op_{ij} is *compatible* with op_{ik} iff $op_{ij} \circ op_{ik}\ (s_i) = op_{ik} \circ op_{ij}\ (s_i)$ for every state s_i of o_j. op_{ij} *conflicts* with op_{ik} $(op_{ij} \rightarrow op_{ik})$ unless op_{ij} is compatible with op_{ik}. If op_{ij} conflicts with op_{ik}, the state obtained by computing op_{ij} and op_{ik} is independent of the computation order. If some operations conflicting with op_i are being computed on o_i, op_i has to wait until the operations complete. In this paper, we assume that "$\rightarrow$" is symmetric, i.e. $op_{ij} \rightarrow op_{ik}$ iff $op_{ik} \rightarrow op_{ij}$. It is written $op_{ij} \leftrightarrow op_{ik}$.

A transaction T manipulates objects $o_1, \ldots, o_n$ by invoking operations $op_1, \ldots, op_n$, respectively. On receipt of the request op_i, o_i is locked in a *lock mode* $\mu(op_i)$. Here, let M_i be a set of lock modes of o_i. For example, *Rlock* and *Wlock* are $\mu(read)$ and $\mu(write)$, respectively. *Dlock* and *Wlock* denote the lock modes of *Deposit* and *Withdraw*.

A mode m_1 is *compatible* with m_2 $(m_1 \rightarrow m_2)$ in M_i if the operations op_1 of mode m_1 is compatible with op_2 of m_2. Otherwise, m_1 conflicts with m_2. For example, two *Rlocks* are compatible in a file object. *Dlock* and *Wlock* are compatible in a *Bank* object. If o_i is locked in a mode m with which $\mu(op_i)$ conflicts, op_i blocks. op_i is computed only if o_i is locked in $\mu(op_i)$. After computing op_i, the lock $\mu(op_i)$ of o_i is released. Problem is when o_i is released. Here, suppose that op_i invokes op_{ij} on o_{ij} $(j = 1, \ldots, l)$. There are the following ways for releasing the locks:

[Releasing schemes]

(1) **Open** : o_i is released when op_i completes.

(2) **Semi-open** : $o_{i1}, \ldots, o_{il}$ are released when op_i completes. However, o_i is not released.

(3) **Close** : Every object locked in op_i is not released. Only if T completes, all the objects locked in T are released. □

In the open scheme, the object o_i is released as soon as op_i completes. Here, the fewest number of the objects are locked. However, the cascading about may occur. On the other hand, every object is locked until the transaction T completes in the close scheme. The close scheme is the same as the strict two-phase locking (2PL) [2]. The largest number of the objects are locked and the throughput of the system is decreased in the close one.

2.2 Replicas

In order to increase the reliability, availability, and performance, an object o_i is replicated in a collection $\{ o_i^1, \ldots, o_i^{k_i} \}$ $(k_i \geq 1)$ of replicas, where o_i^j is a replica of o_i. Each o_i^j supports the same data and operations as the other replicas. That is the objects are assumed to be fully replicated. Let $r(o_i)$ be $\{ o_i^1, \ldots, o_i^{k_i} \}$ $(k_i \geq 1)$.

Here, suppose that an operation op_i on an object o_i is invoked. Suppose that op_i invokes an operation op_{ij} on an object o_{ij} and o_{ij} further invokes op_{ijk} on o_{ijk} [Fig. 1]. Here, suppose that $r(o_{ij}) = \{ o_{ij}^1, \ldots, o_{ij}^{k_{ij}} \}$. If op_i sends a request op_{ij} to the replicas $o_{ij}^1, \ldots, o_{ij}^{k_{ij}}$, op_{ij} is computed on the replicas. On receipt of op_{ij}, o_{ij}^k computes op_{ij} and then invokes op_{ijk}. Since multiple replicas invoke op_{ijk}, op_{ijk} is computed multiple times on o_{ijk}. For example, if op_{ijk} is an operation to add some value x, to value of o_{ijk} is incremented by $k_{ij} \cdot x$, not x. Thus, If op_{ijk} updates o_{ijk}, the state of o_{ijk} gets inconsistent since op_{ijk} is computed multiple times on o_{ijk}. This is named *inconsistent* redundant invocation [11]

In order to resolve the *inconsistent* redundant invocations of the operations on replicas, the following invocation rule is adopted. There are two cases:

(1) op_{ij} does not invoke any operation; If op_{ij} changes the state of o_{ij}, op_{ij} is computed on every replica o_{ij}^k. Otherwise, op_{ij} is computed in one replica o_{ij}^k.

(2) op_{ij} invokes some operations op_{ijk} on o_{ijk}; op_{ij} is invoked on one replica o_{ij}^k or every replica if op_{ij} changes o_{ij} or not like (1). In either case, only one replica o_{ij}^k invokes op_{ijk}. On receipt of the response of op_{ijk} from o_{ijk}, o_{ij}^k forwards it to all the other replicas if op_{ij} changes o_{ij}^k. On receipt of the state for o_{ij}^k, o_{ij}^h $(h \neq k)$ changes the state so as to be the same as o_{ij}^k by using the state.

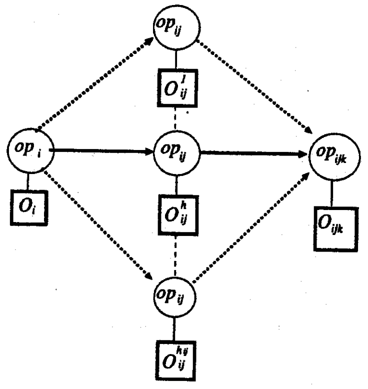

Figure 1: Invocation on replicas.

3 Optimistic Object-Based Locking

The system includes multiple objects $o_1, \ldots, o_n$. Each object o_i supports a set π_i of operations $op_{i1}, \ldots, op_{il_i}$ ($l_i \geq 1$). o_i is replicated in a set $r(o_i)$ of replicas $o_i^1, \ldots, o_i^{k_i}$ ($k_i \geq 1$). We discuss an *optimistic object-based locking* (*OOBL*) protocol for maintaining the consistency among the replicas of o_i.

3.1 Lock modes

First, we discuss how the lock modes supported by the object o_i are related. Before computing op_i, o_i is locked in a mode $\mu(op_i)$ in M_i. Suppose that o_i is locked in a mode m and op_i would be computed on o_i. If $\mu(op_i)$ is *compatible* with m, op_i can be started to be computed on o_i. Otherwise, op_i has to wait until the lock of the mode m is released. Here, let $C_i(m)$ be a set of modes with which m conflicts in o_i, i.e. $C_i(m) = \{m' \mid m'$ conflicts with $m\}$. In this paper, we assume that the compatibility relation among the modes are symmetric. Hence, m is in $C_i(m')$ for every m' in $C_i(m)$.

[Definition] For every pair of modes m_1 and m_2 in M_i, m_1 is *more restricted* than m_2 ($m_1 \Rightarrow m_2$) iff $C_i(m_1) \supseteq C_i(m_2)$.□

[Example 1] Let us consider a file object f with operations *read* and *write*. Let *Rlock* and *Wlock* be lock modes for *read* and *write*, respectively. Since *read* conflicts with *write*, *Rlock* conflicts with *Wlock*. *Rlock* is compatible with *Rlock*. $C_f(Rlock) = \{Wlock\}$ and $C_f(Wlock) = \{Wlock, Rlock\}$. Since $C_f(Rlock) \subseteq C_f(Wlock)$, *Wlock* is more restricted than *Rlock* (*Wlock* $\Rightarrow$ *Rlock*). □

[Example 2] A *Bank* object b supports abstract operations *Deposit*, *Withdraw*, and *Check*. Let *Dlock*, *Wlock*, and *Clock* be lock modes $\mu(Deposit)$, $\mu(Withdraw)$, and $\mu(Check)$. *Dlock* and *Wlock* are compatible. *Clock* conflicts with *Dlock* and *Wlock*. $C_b(Dlock) = C_b(Wlock) = \{Clock\}$ and $C_b(Clock) = \{Dlock, Wlock\}$. Since $C_b(Dlock) \cap C_b(Clock) = \phi$, neither *Dlock* $\Rightarrow$ *Clock* nor *Clock* $\Rightarrow$ *Dlock*. □

In Example 2, there is no restriction relation $\Rightarrow$ among the modes. However, *Clock* has more modes conflicting with *Clock* than *Dlock* and *Wlock*. We discuss which modes are stronger.

[Definition] For every pair of modes m_1 and m_2 in M_i, m_1 is *stronger* than m_2 iff $m_1 \in C_i(m_2)$ and $|C_i(m_1)| \geq |C_i(m_2)|$. □

In Example 2, $|C_b(Clock)| > |C_b(Dlock)|$. Here, *Clock* is stronger than *Dlock* and *Wlock*. The stronger the lock mode m is, the more conflicting modes m has. It is straightforward that $|C_i(m_1)| \geq |C_i(m_2)|$ if $m_1 \Rightarrow m_2$.

Some mode may be more frequently used than others. If a mode m_2 conflicting with a mode m_1 is used frequently, an operation of m_1 has to wait more often. Here, let $\varphi(m)$ be the usage frequency of a mode m, i.e. how many operations whose modes are m are issued to o_i for a unit time. The frequencies of the modes in o_i are normalized to be $\sum_{m \in M_i} \varphi(m) = 1$. The *weighted strength* $\|C_i(m)\|$ is defined to be $\sum_{m' \in C_i(m)} \varphi(m')$.

[Definition] For every pair of modes m_1 and m_2 in M_i, m_1 is *stronger* than m_2 on the *weight* (m_1 is *w-stronger* than m_2 : $m_1 \succeq m_2$) iff $m_1 \in C_i(m_2)$, $m_2 \in C_i(m_1)$, and $\|C_i(m_1)\| \geq \|C_i(m_2)\|$. □

The more *w-stronger* a mode m is, the longer an operation of m has to wait.

It is clear that $\|C_i(m_1)\| \geq \|C_i(m_2)\|$ if $m_1 \Rightarrow m_2$. Hence, $m_1 \succeq m_2$ if $m_1 \Rightarrow m_2$. Suppose that op_1 and op_2 are operations of modes m_1 and m_2 in an object o_i, respectively. Let us consider a *blocking* probability that op_i waits for the release of the lock conflicting with op_i. If $\|C_i(m_1)\| \geq \|C_i(m_2)\|$, op_1 has a higher blocking probability than op_2.

In Example 1, *Wlock* $\succeq$ *Rlock* since *Wlock* $\Rightarrow$ *Rlock*. In Example 2, suppose that the usage ratios of *Clock*, *Dlock*, and *Withdraw* are 60%, 30%, and 10%, respectively, i.e. $\varphi(Clock) = 0.6$, $\varphi(Dlock) = 0.3$, and $\varphi(Wlock) = 0.1$. $\|C_b(Clock)\| = \varphi(Dlock) + \varphi(Wlock) = 0.4$. $\|C_b(Dlock)\| = \|C_b(Wlock)\| = \varphi(Clock) = 0.6$. Hence, *Clock* $\preceq$ *Dlock* and *Clock* $\preceq$ *Wlock* since $\|C_b(Clock)\| < \|C_b(Dlock)\|$ and $\|C_b(Clock)\| \leq \|C_b(Wlock)\|$. *Dlock* and *Wlock* have a higher blocking probability than *Clock*.

The modes in M_i are partially ordered by the *w-strength* relation "$\preceq$". A mode m is referred to as *maximal* in M_i iff there is no mode m' in M_i such that $m \preceq m'$. m is *maximum* iff $m' \preceq m$ for every mode m' in M_i. The *minimal* and *minimum* modes are defined in the similar way. For every pair of modes m_1 and m_2 in M_i, a mode m is a *least upper bound* (*lub*) $m_1 \cup m_2$ of modes m_1 and m_2 iff (1) $m_1 \preceq m$, (2) $m_2 \preceq m$, and (3) there is no mode m' such that $m_1 \preceq m' \preceq m$ and $m_2 \preceq m' \preceq m$. The *greatest lower bound* (*glb*) $m_1 \cap m_2$ is similarly defined.

3.2 Equivalent class

Objects can generally support arbitrary number of operations while files support only *read* and *write*. The more operations the objects support, the more complex it is to analyze the strength relation among the operations. Hence, we first try to reduce the number of operations to be analyzed.

First, we partition the set π_i of operations of

o_i into disjoint groups which are composed of operations related.

[Definition] For every pair of operations op_1 and op_2 in π_i, op_1 and op_2 are *related* ($op_1 \sim op_2$) iff one of the following conditions holds;

(1) op_1 and op_2 conflict.

(2) $op_1 \sim op_3$ and $op_3 \sim op_2$ for some operation op_3 in π_i. □

Here, "$\sim$" is reflexive and symmetric. Hence, "$\sim$" is equivalent. π_i is partitioned into the equivalent classes by using "$\sim$". Here, let $R_i(op_1)$ denote an equivalent class $\{op_2 \mid op_1 \sim op_2 \text{ in } \pi_i\}$ of op_1, i.e. for every op_2 in $R_i(op_1)$, $R_i(op_1) = R_i(op_2)$. op_i and op_2 are compatible in o_i. if $R_i(op_1) \neq R_i(op_2)$, i.e. op_1 and op_2 are not related. It is noted that $op_1 \sim op_2$ may hold even if op_1 and op_2 are compatible.

Let us consider an example where an object o_i supports six operations $op_1, \ldots, op_6$ in π_i. Suppose that there is a conflicting relation, i.e. $op_1 \leftrightarrow op_4$ (op_1 conflicts with op_4), $op_2 \leftrightarrow op_5$, $op_2 \leftrightarrow op_6$, $op_3 \leftrightarrow op_5$ and $op_3 \leftrightarrow op_6$ [Fig. 2]. $R_{i1} = R_i(op_1) = R_i(op_4) = \{op_1, op_4\}$. $R_{i2} = R_i(op_2) = R_i(op_3) = R_i(op_5) = R_i(op_6) = \{op_2, op_3, op_5, op_6\}$. π_i is partitioned into two equivalent classes R_{i1} and R_{i2} as shown in Fig. 2. Since operations in different classes are compatible with each other, we can discuss the strength of the lock mode in one class independently of the other classes.

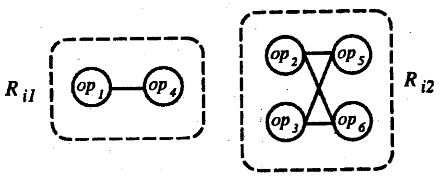

Figure 2: Related operations.

Here, suppose that there is an equivalent class R_{ij} in o_i ($j = 1, \ldots, t_i$).

[Definition] For every pair of operations op_1 and op_2 in each equivalent class R_{ij}, op_1 and op_2 are *at the same level* ($op_1 \equiv op_2$) iff op_1 and op_2 are compatible and $C_i(op_1) = C_i(op_2)$. □

In Fig. 2, op_2 and op_3 are at the same level ($op_2 \equiv op_3$) since op_2 and op_3 are compatible and $C_i(op_2) = C_i(op_3) = \{op_5, op_6\}$. $op_5 \equiv op_6$. op_5 and op_6 are not at the same level as op_2.

Here, let $\varphi(op_{ij})$ denote a usage frequency that op_{ij} is invoked in o_i. In this paper, we assume that every pair of operations op_1 and op_2 have different lock mode, i.e. $\mu(op_1) \neq \mu(op_2)$. Hence, $\varphi(op_{ij}) = \varphi(\mu(op_{ij}))$. The frequencies are normalized to be $\sum_{op \in R_i(op_{ij})} \varphi(op) = 1$ for each equivalent class $R_{ij}(op)$. Each equivalent class R_{ij} can be reduced as follows :

[Reduction]

(1) Let $S_{ij}(op_{ij})$ be a set of operations which are at the same level op_{ij} in R_{ij}, i.e. $S_{ij}(op_{ij}) = \{op' \mid op_{ij} \equiv op'\}$.

(2) For each operation op_{ij} in R_i, all the operations in $S_{ij}(op_{ij})$ are replaced with a virtual operation op.

(3) $\varphi(op)$ is given as $\sum_{op' \in S_{ij}(op_{ij})} \varphi(op')$. □

[Example 3] The graph R_{i2} shown in Fig. 2 can be reduced to one shown in Fig. 3. Since $op_2 \equiv op_3$, op_2 and op_3 are merged into one virtual operation $op_{2,3}$. op_5 and op_6 are also merged into $op_{5,6}$ since $op_5 \equiv op_6$. Suppose that $\varphi(op_2) = 0.4$, $\varphi(op_3) = 0.3$, $\varphi(op_5) = 0.2$, and $\varphi(op_6) = 0.1$ in R_{i2}. Here, $\varphi(op_{2,3}) = \varphi(op_2) + \varphi(op_3) = 0.7$ and $\varphi(op_{5,6}) = 0.3$. $\varphi(op_1) = 0.6$ and $\varphi(op_4) = 0.4$ in R_{i2}. Since $R_{i1} = \{op_1, op_4\}$ and $R_{i2} = \{op_2, op_3, op_5, op_6\}$, $\varphi(op_1) + \varphi(op_4) = 1.0$ and $\varphi(op_2) + \varphi(op_3) + \varphi(op_5) + \varphi(op_6) = 1.0$.□

From here, we assume that we consider each reduced class in an object.

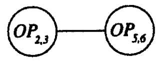

Figure 3: Merged operations.

3.3 Locking protocol

The traditional systems support has only two types of operations, i.e. *read* and *write*, where *Wlock* is stronger than *Rlock*. In the O2PL, the replicas of the object to be written are first locked in *Rlock*. The replicas are finally locked in *Wlock*. In the object-based system, each object o_i supports more abstract objects which support more types of operations. That is, there may be more than two lock modes. For example, the *Bank* object b supports *Check*, *Deposit*, *Withdraw*, and *Transfer* operations. In addition, the lock modes in M_i are partially ordered by the strength relation "$\preceq$" as discussed in the preceding subsection.

We discuss the optimistic concurrency control to maintain the mutual consistency among multiple replicas $o_i^1, \ldots, o_i^{k_i}$ of an object o_i. Suppose that a transaction T invokes an operation op_{ij} on o_i. First, some number of the replicas in $r(o_i)$ are locked in a mode $\mu_1(op_{ij})$ which is not stronger than $\mu(op_{ij})$. Let $f_1(op_{ij})$ ($\leq k_i$) be the number of the replicas locked by op_{ij} before op_{ij} is computed. Unless all of $f_1(op_{ij})$ replicas can be successfully locked, op_{ij} is aborted. If all of $f_1(op_{ij})$ replicas could be locked, op_{ij} is computed on the replicas as presented before. When T would commit, some number $f_2(op_{ij})$ of the replicas are locked in a mode $\mu(op_{ij})$. $f_1(op_{ij}) \leq f_2(op_{ij})$ and $\mu_1(op_{ij}) \preceq \mu(op_{ij})$.

We discuss how to decide the numbers $f_1(op_{ij})$ and $f_2(op_{ij})$ of the replicas to be locked and the lock mode $\mu_1(op_{ij})$. The more replicas are locked, the more communication and computation are required. Hence, the more frequently op_{ij} is invoked, the fewer replicas are locked. $f_1(op_{ij})$ and $f_2(op_{ij})$ are decided depending on the probability that op_{ij} conflicts with other operations of o_i. There are the following constraints on f_1 and f_2:

(1) If $\mu(op_{ij})$ conflicts with $\mu(op_{ik})$, $f_2(op_{ij}) + f_2(op_{ik}) > k_i$ and $f_2(op_{ij}) + f_1(op_{ik}) > k_i$.

(2) $f_2(op_{ij}) \geq f_2(op_{ik})$ iff $\|C_i(op_{ij})\| \geq$

$||C_i(op_{ik})||$.

Here, suppose that op_{ij} locks $f_h(op_{ij})$ replicas in $r(o_i) = \{o_i^1, \ldots, o_i^{k_i}\}$. Suppose that two operations op_{ij} and op_{ik} are invoked and conflict in o_i. The probability that both op_{ij} and op_{ik} can lock the required number of the replicas is given 1 - [$f_s(op_{ij}) \cdot \varphi(op_{ij})/k_i$] [$f_h(op_{ik}) \cdot \varphi(op_{ik})/k_i$]. $C_i(op_{ij})$ is a set of operations conflicting with op_{ij} in o_i. Here, the probability $F_h(op_{ij})$ that op_{ij} can lock $f_h(op_{ij})$ replicas is 1 - $\prod_{op \in C_i(op_{ij})}$ $[f_h(op) \cdot \varphi(op)/k_i]$. $f_2(op_{ij})$ can be decided so that the probability $F_2(op_{ij})$ is the minimum.

In this paper, we present a simplified *OOBL* protocol, where $f_1(op_{ij})$ is equal to $f_2(op_{ij})$, say $f(op_{ij}) = f_1(op_{ij}) = f_2(op_{ij})$, and $\varphi(op_{ij})$ is the same for every operation op_{ij} of the object o_i. We first define a *bottom* mode $\perp$ for every class R_i of o_i. $\perp$ is defined to be a mode satisfying the constraint that $\perp \preceq m$ for every mode m in R_i and $\perp$ is compatible with $\perp$. $\perp$ is the lub of M_i. If $M_i = \{Rlock, Wlock\}$, $\perp$ is *Rlock* because *Rlock* $\preceq$ *Wlock*. Unless there is the minimum in M_i, an artificial mode $\perp$ is included in M_i.

Suppose that an operation op_{ij} in a transaction T would lock replicas $o_i^1, \ldots, o_i^{k_i}$ of o_i. o_i is locked by the following locking scheme.

[Locking scheme]

(1) First, $f(op_{ij})$ replicas in $r(o_i)$ are randomly selected. Let S_i be a subset of the replicas selected from $r(o_i)$.

(2) Every replica in S_i is locked in the bottom mode $\perp$.

(3) If succeeded in locking all the replicas in S_i, the replicas in S_i are manipulated by op_{ij}.

(4) When the transaction T commits, the lock mode of every replica in S_i is converted up to $\mu(op_{ij})$. If succeeded, T commits. Otherwise, T aborts. □

4 Concluding Remark

This paper has discussed the *optimistic object-base locking* (*OOBL*) protocol on the replicas of the objects. The objects support more abstract operations than the traditional *read* and *write* operations. We have defined to weighted strength of lock modes based on the conflicting relation and usage frequency. In the *OOBL* protocol, the number of replicas to be locked is decided based on the weighted strength of the lock modes.

References

[1] Barbara, D. and Imielinski, T., "Sleepers and Workaholics: Caching Strategies in Mobile Environments," *Proc. of the ACM SIGMOD* , pp. 1–12, 1994.

[2] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems," *Addison - Wesley* , 1987.

[3] Carey, J. M. and Livny, M., "Conflict Detection Tradeoffs for Replicated Data," *ACM TODS* , Vol. 16, No.4, pp. 703-746, 1991.

[4] Ellis, C. A., Gibbs, S. J., and Rein, G. L., "Groupware," *Comm. ACM* , Vol.34, No.1, pp.38-58, 1991.

[5] Gruber, R., Kaashoek, F., Liskov, B., and Shrira, L., "Disconnected Operation in the Thor Object-Oriented Database System," *Proc. of IEEE Workshop on Mobile Computing Systems and Applications* , pp. 51–56, 1994.

[6] Jing, J., Bukhres, O., and Elmagarmid, A., "Distributed Lock Management for Mobile Transactions," *Proc of IEEE ICDCS* – 15, pp. 118-125, 1995.

[7] Kistler, J. J. and Satyanaranyanan, M., "Disconnected Operation in the Coda File System," *ACM Trans. on Database Systems*, Vol. 10, No. 1, pp. 2–25, 1992.

[8] Kung, H. T. and Robinson, J. T., "On Optimistic Methods for Concurrency Control," *ACM Trans. on Database Systems*, Vol.6, No.2, pp. 81-87, 1994.

[9] Moss, J. E., "Nested Transactions : An Approach to Reliable Distributed Computing," *The MIT Press Series in Information Systems*, 1985.

[10] Silvano, M. and Douglas C, S., "Constructing Reliable Distributed Communication Systems with CORBA" *IEEE Communications Magazine*, Vol.35, No.2, pp.56–60, 1997.

[11] Yoshida, T. and Takizawa, M., "Model of Mobile Objects, " *Proc. of the 7th DEXA* (*Lecture Notes in Computer Science, No 1134, Springer-Verlag*), pp. 623-632, 1996.