Object-Based Quorum Scheme for Replicated Objects

Kyouji Hasegawa, Hiroaki Higaki, and Makoto Takizawa
Dept. of Computers and Systems Engineering, Tokyo Denki University
Email {kyo, hig, taki}@takilab.k.dendai.ac.jp

Abstract

In object-based systems, objects are replicated to increase the performance, reliability, and availability. We discuss a novel object-based locking (OBL) protocol to lock replicas of objects by extending the quorum-based protocol for read and write to abstract methods. Unless two methods conflict, subsets of the replicas locked by the methods do not intersect even if the methods change the replicas. Methods not computed on a replica A but computed on another replica are computed on A when a method conflicting with the methods are issued to A in the OBL protocol. We newly propose a version vector to identify what methods are computed on a replica.

1 Introduction

Reliable distributed systems are composed of replicas of objects. The replicas of the object have to be mutually consistent. The two-phase locking (2PL) protocol [1,2] locks one of the replicas for *read* and all the replicas for *write*. The 2PL protocol is not efficient for write-dominated applications because all the replicas are locked for *write*. In the quorum-based protocol [3], some numbers Q_r and Q_w of the replicas named *quorum numbers* are locked for *read* and *write*, respectively. Here, a constraint " $Q_r + Q_w > a$ " for the number a of the replicas has to be satisfied.

Distributed applications are modeled to be a collection of multiple objects which are cooperating. Object-based frameworks like CORBA [7] are widely used to develop the distributed applications. An object supports abstract methods like *Deposit* in a *bank* object. The object is locked in an abstract mode corresponding to a method. A pair of methods op_1 and op_2 supported by an object o conflict if the result obtained by applying op_1 and op_2 to o depends on the computation order of op_1 and op_2 [1]. In this paper, we propose a novel locking scheme for replicated objects named OBL (*object-based locking*) protocol, which is an extension of the quorum-based protocol [3] to the replicas of the abstract objects. Before computing a method op_t on an object o, some quorum number Q_t of the replicas of o are locked in the abstract mode for op_t. Q_t depends on how *frequently* op_t is invoked. The more frequently op_t is invoked, the smaller Q_t gets. Suppose a pair of methods op_t and op_u are issued to replicas of the object o. If op_t and op_u are update methods, the quorum-based protocol requires that " $Q_t + Q_u > a$ " hold. If op_t and op_u are computed on a pair of replicas o^t and o^u, respectively, the states of o^t and o^u get different. Then, if both op_t and op_u are computed on replicas o^t and o^u, respectively, o^t and o^u get the same state if op_t and op_u are compatible. In the quorum-based protocol, there must be at least one newest replica where all write methods issued are computed. However, there can exist replicas from which the newest version can be constructed even if there is no newest replica. In order to do that, we have to identify what methods are computed on each replica. We newly propose a *version vector* to identify the methods computed on each replica. In the OBL protocol, fewer number of replicas are locked than the quorum-based protocol and the $2PL$ protocol.

In section 2, we present the system model. In section 3, we extend the quorum concepts to object-based system. In section 4, we discuss the OBL protocol with the version vector. In section 5, we evaluate the OBL protocol compared with the quorum-based protocol in terms of the number of replicas to be locked.

2 Objects

A system is composed of objects $o_1, \ldots, o_n$ ($n \geq 1$) which are cooperating by exchanging messages in a reliable network. Each object o_i supports methods $op_{i1}, \ldots, op_{il_i}$ ($l_i \geq 1$) for manipulating o_i and is encapsulated so that o_i can be manipulated only through the methods supported by o_i. By using the network, o_i can send messages to o_j with no message loss in the sending order.

The objects are distributed on multiple processors. Objects are stored in *server* processors. If a transaction in a *client* processor sends a request message of a method op_i to an object o_i in a server, o_i computes op_i. Here, op_i may invoke a method op_{ij} on another object o_{ij}. The methods are nested. o_i sends the response of op_i back to the transaction. A transaction is an atomic invocation sequence of methods. The transaction an method *commit* only if all the methods invoked commit. A method which changes the state of the object is an *update* one.

Let $op(s)$ denote a state obtained by applying a method op to a state s of an object o_i. A method op_{ij} is *compatible* with another method op_{ik} iff $op_{ij} \circ op_{ik}(s_i) = op_{ik} \circ op_{ij}(s_i)$ for every state s_i of o_i. op_{ij} *conflicts* with op_{ik} unless op_{ij} is compatible with op_{ik}. The conflicting relation C_i among the methods is not transitive. We assume C_i is symmetric. C_i is assumed to be specified on definition of the object o_i. The

interleaved and parallel computation of methods has to be *serializable* [1].

On receipt of a request of a method op_i, an object o_i is locked in a *lock mode* $\mu(op_i)$ in order to make the computation serializable. If op_1 is compatible with op_2, the mode $\mu(op_1)$ is compatible with $\mu(op_2)$. Otherwise, the mode $\mu(op_1)$ conflicts with $\mu(op_2)$. After computing op_i, the lock of the mode $\mu(op_i)$ on the object o_i is released. Let $M_i(m)$ be a set of lock modes with which a lock mode m conflicts in o_i.

3 Object Quorums

3.1 Quorum constraint

Let $R(o_i)$ be a *cluster* of an object o_i, i.e. a set of replicas $o_i^1, \ldots, o_i^{a_i}$ of o_i ($a_i \geq 1$). We extend the quorum-based protocol [3] to lock the replicas in the object-based systems. Let N_{it} be a subset of replicas to be locked by a method op_{it}, named a *quorum set* of op_{it} ($N_{it} \subseteq R(o_i)$). Let Q_{it} be the *quorum* number of the replicas in N_{it}. The quorum numbers have to satisfy the following object-based locking (OBL) constraint.

[OBL constraint]

- If $\mu(op_{it})$ conflicts with $\mu(op_{iu})$, $Q_{it} + Q_{iu} > a_i$.

Let $\varphi(op_{it})$ be usage frequency of a method op_{it}, i.e. how frequently op_{it} is issued to o_i. Here, $\varphi(op_{i1}) + \ldots + \varphi(op_{il_i}) = 1$. $Q_{i1}, \ldots, Q_{il_i}$ are obtained so as to minimize the average number $\varphi(op_{i1})Q_{i1} + \ldots + \varphi(op_{il_i})Q_{il_i}$ of the replicas locked. This means, the more frequently op_{it} is invoked, the fewer number of the replicas are locked by op_{it}.

A transaction T locks the replicas $o_i^1, \ldots, o_i^{a_i}$ as follows before manipulating the replicas by op_{it}. First, a quorum set N_{it} is fixed for op_{it} in the cluster $R(o_i)$. Every replica in N_{it} is locked in a mode $\mu(op_{it})$. If all the replicas in N_{it} are locked, the replicas in N_{it} are manipulated by op_{it}. When T commits, the locks on the replicas in N_{it} are released.

According to the quorum-based protocol, $Q_{it} + Q_{iu} > a_i$ if op_{it} and op_{iu} are update methods. Here, a_i is the number of the replicas of an object o_i. In the OBL protocol, $Q_{it} + Q_{iu} > a_i$ only if op_{it} conflicts with op_{iu}. In another word, $Q_{it} + Q_{iu} > a_i$ can hold even if op_{it} or op_{iu} is an update method. The OBL protocol satisfies the following properties.

[Properties] For every pair of conflicting methods op_{it} and op_{iu} of an object o_i:

1 At least $Q_{it} + Q_{iu} - a_i$ replicas compute both of the methods op_{it} and op_{iu}.

2 If a pair of replicas o_i^h and o_i^k compute both op_{it} and op_{iu}, o_i^h and o_i^k compute op_{it} and op_{iu} in the same order. □

3.2 Precedency among replicas

[Example 1] Each of replicas B^1, B^2, B^3, and B^4 of the bank object B supports four methods *Deposit* (D), *Withdraw* (W), *Check* (C), and *Audit* (A). The method D is compatible with the method W. C is compatible with A. D and W conflict with C and A. D, W, and A are update ones but C is not. Figure 1 indicates a graph showing the conflicting relation among the methods. Here, a node shows a method and an edge between the nodes indicates that the nodes conflict. Each replica B^i has a version number V^i whose initial value is 0. Let the quorum number Q_D of D be 3 and Q_W of W be 2. Here, $Q_D + Q_W > 4$. D is issued to three replicas, say B^1, B^2, and B^3 and $V^1 = V^2 = V^3 = 1$. Then, W is issued to B^1 and B^4. Since $V^1(=1) > V^4(=0)$, W is computed on B^1 and $V^1 = 2$. B^4 is updated by taking the state from B^1. Here, $V^1 = V^4 = 2$ and $V^2 = V^3 = 1$. If the quorum number is decided based on the conflicting relation, we can reduce the quorum number but cannot decide which replica is the newest by using the version numbers. Here, $Q_{it} + Q_{iu} > a_i$ if a pair of methods op_{it} and op_{iu} conflict. For example, Q_D, Q_W, Q_C, and Q_A can be 2, 2, 3, and 3, respectively. First, suppose D is issued to B^1 and B^2 and W is issued to B^3 and B^4 since $Q_D = Q_W = 2$. Here, the version numbers of the replicas are changed to 1, i.e. $V^1 = V^2 = V^3 = V^4 = 1$. Then, C is issued to B^1, B^2, and B^3 since $Q_C = 3$. Here, B^3 is different from B^1 and B^2 although they have the same version number 1. Since D and W are compatible, the instance of D computed on B^1 and B^2 is also computed on B^3 and B^4 and the instance of W computed on B^3 and B^4 is computed on B^1 and B^2. Then, C can be computed on one of the replicas, say B^1. Thus, the replicas can be the newest by computing methods which are not computed on the replicas but computed on the others if these methods are compatible. Problem is that the states of B^1 and B^2 cannot be recognized to be different from B^3 and B^4 by using the version numbers because the version numbers of B^1, B^2, B^3, and B^4 are 1 after D and W are computed. □

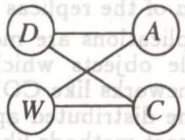

Figure 1: Conflicting graph.

A replica o_i^h is considered to be *newer* than another replica o_i^k if every method instance computed on o_i^k is computed on o_i^h.

[Definition] A replica o_i^h *precedes* another replica o_i^k ($o_i^h \rightarrow o_i^k$) iff every update method computed on o_i^k is computed on o_i^h. □

" $o_i^h \rightarrow o_i^k$ " means that o_i^k is obsolete since some update methods computed on o_i^h are not computed on o_i^k. o_i^h is *maximal* iff there is no replica o_i^k such that $o_i^h \rightarrow o_i^k$ in the cluster $R(o_i)$. o_i^h is *maximum* iff $o_i^k \rightarrow o_i^h$ for every replica o_i^k in $R(o_i)$. The quorum-based protocol requires that every quorum set include at least one maximum replica o_i^h. That is, if a pair of

update methods op_{it} and op_{iu} are issued, at least one replica o_i^h computes both op_{it} and op_{iu}. In the *OBL* protocol, no replica may compute both of op_{it} and op_{iu} if op_{it} and op_{iu} are compatible even if op_{it} and op_{iu} are update ones. Hence, there may be no maximum replica but may exist multiple maximal replicas. $R(o_i)$ is *complete* iff there is a maximum replica in $R(o_i)$. $R(o_i)$ may be incomplete in the *OBL* protocol.

[Definition] A pair of maximal replicas o_i^h and o_i^k are *unifiable* iff o_i^h and o_i^k get the same state if every update method not computed on one of o_i^h and o_i^k is computed on the other replica. □

Suppose there are two replicas o_i^h and o_i^k. Let $\langle op_{h1}, \ldots, op_{hl_h} \rangle$ be a sequence π_{hk} of update methods computed on a replica o_i^h but not on o_i^k. Let $\langle op_{k1}, \ldots, op_{kl_k} \rangle$ be a sequence π_{kh} of update methods computed on o_i^k but not on o_i^h. If every pair of methods op_{hu} and op_{kv} are compatible for $u = 1, \ldots, l_h$ and $v = 1, \ldots, l_k$, a state obtained by applying π_{hk} to o_i^k is the same as a state obtained by applying π_{kh} to o_i^h. The state obtained here is a *least upper bound* of o_i^h and o_i^k ($o_i^h \cup o_i^k$) with respect to "$\rightarrow$". For example, suppose B^1 and B^2 compute D and B^3 and B^4 compute W in Example 1. Here, $\pi_{13} = \pi_{23} = \langle W \rangle$ and $\pi_{31} = \pi_{41} = \langle D \rangle$. The unifiable relation "$\equiv$" is equivalent. Let $U(o_i^h)$ be an equivalent set $\{ o_i^k \mid o_i^k \equiv o_i^h$ in $R(o_i) \}$ for a maximal replica o_i^h. A cluster $R(o_i)$ is *consistent* iff $U(o_i^h) = U(o_i^k)$ for some pair of maximal replicas o_i^h and o_i^k in $R(o_i)$. Here, $U(o_i^h)$ is referred to as *unifiable* set $U(o_i)$ of $R(o_i)$. A least upper bound of the replicas in a consistent cluster $R(o_i)$ shows a possible maximum replica to be obtained from the replicas in $R(o_i)$. In the quorum-based protocol, there exists a maximum replica. If $R(o_i)$ is inconsistent, the replicas cannot be consistent.

Incomplete methods are update ones computed on some replicas but not on every replica in a unifiable set $U(o_i)$ of $R(o_i)$. Every pair of incomplete methods not computed on a same replica are computed not on every replica. Complete methods are update methods computed on every replica in $U(o_i)$.

Let us consider how op_{it} is computed on the replicas. op_{it} can be computed on a replica o_i^h in the quorum set N_{it} if every incomplete method which conflicts with op_{it} is computed on o_i^h. However, there might not exist such a replica o_i^h in $R(o_i)$. Hence, op_{it} is computed as follows.

1 Incomplete methods on each maximal replica are computed on the other maximal replicas in N_{it} as presented before. Here, every maximal replica is the newest one.
2 Then, op_{it} is computed on the maximal replicas.
3 If op_{it} is an update method, the states of the replicas in N_{it} have to be changed. The non-maximal replicas in $R(o_i)$ compute every update method computed on the maximal ones but not computed on the replicas. In another way, one of the maximal replicas sends the state to the other replicas. Here, every replica in N_{it} is the newest one, i.e. maximum replica in $R(o_{it})$.

3.3 Version vector

We newly introduce a *version vector* to identify what methods are computed on each replica. Each replica o_i^h manipulates a bitmap vector $BM_i^h = \langle BM_{i1}^h, \ldots, BM_{il_i}^h \rangle$ and a counter vector $U_i^h = \langle U_{i1}^h, \ldots, U_{il_i}^h \rangle$. Each element BM_{it}^h is in a bitmap form $\langle BM_{it}^{h1}, \ldots, BM_{it}^{ha_i} \rangle$. The kth bit BM_{it}^{hk} is 1 if o_i^h knows that op_{it} is issued to o_i^k, otherwise 0 ($k = 1, \ldots, a_i$). Each element U_{it}^h is a *version number* of the replica o_i^h with respect to op_{it}. U_{it}^h is incremented by one each time op_{it} is computed on o_i^h and op_{it} is an update method. For example, BM_B^i and U_B^i of a replica B^i are $\langle BM_{BD}^i, BM_{BW}^i, BM_{BC}^i, BM_{BA}^i \rangle$ and $\langle U_{BD}^i, U_{BW}^i, U_{BC}^i, U_{BA}^i \rangle$, respectively, in Example 1 [Figure2]. Here, let V_{it}^h denote $(U_{it}^h)_{BM_{it}^h}$. For example, $V_{BD}^2 = 3_{1101}$ shows $BM_{BD}^2 = 1101$ and $U_{BD}^2 = 3$. V_i^h is a *version vector* $\langle V_{i1}^h, \ldots, V_{il_i}^h \rangle$. o_i^h manipulates the version vector V_i^h which includes both information on BM_i^h and U_i^h.

[Example 2] Initially, each replica B^j has the version vector $V_B^j = \langle 0_{0000}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$ for $j = 1, \ldots, 4$ in Example 1. Suppose D is issued to B^1 and B^2 since $Q_D = 2$. $V_B^1 = V_B^2 = \langle 1_{1100}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$. Then, W is issued to B^3 and B^4 since $Q_W = 2$. $V_B^3 = V_B^4 = \langle 0_{0000}, 1_{0011}, 0_{0000}, 0_{0000} \rangle$. Then, C is issued to B^1, B^2, and B^3 since $Q_C = 3$. $V_B^1 = V_B^2 \neq V_B^3$. Since $V_{BD}^1 = V_{BD}^2 = 1_{1100}$ and $V_{BW}^3 = 1_{0011}$, B^1 and B^2 know that D is issued to B^1 and B^2, and B^3 knows that W is issued to B^3 and B^4. Here, no replica is maximum because every replica has computed either D or W. One replica, say B^1, is selected. The instance of W computed on B^3 is computed on B^1. V_{BW}^1 is changed to be 1_{0011}, i.e. $V_B^1 = \langle 1_{1100}, 1_{0011}, 0_{0000}, 0_{0000} \rangle$. Then, C is computed on B^1. Since C is not an update one, V_{BC}^1 is not changed. D is issued to B^3 and B^4. $V_B^3 = V_B^4 = \langle 1_{0011}, 1_{0011}, 0_{0000}, 0_{0000} \rangle$. Then, the method A is issued to three replicas B^2, B^3, and B^4 since $Q_A = 3$. Since $V_{BD}^2 = 1_{1100}$ and $V_{BD}^3 = 1_{0011}$, B^2 and B^3 compute different instances D_1 and D_2 of D, respectively. B^2 and B^3 exchange the instances of D_1 and D_2. In addition, $V_{BW}^2 = 0_{0000}$ and $V_{BW}^3 = 1_{0011}$. Here, the instances of D and W computed on B^3 have to be computed on B^2 to obtain the least upper bound version of B^2 and B^3. Since D is compatible with W, D and W can be computed on B^2 in any order. Now, the method A is computed on B^2 after D and W are computed. Here, B^2 computes a sequence $D_1 W D_2 A$ and B^2 computes $W D_2 D_1 A$. $V_B^2 = \langle 2_{1111}, 1_{0011}, 0_{0000}, 1_{0111} \rangle$ and A updates B^2. If the state of B^2 is sent to B^3 and B^4, B^3 and B^4 are

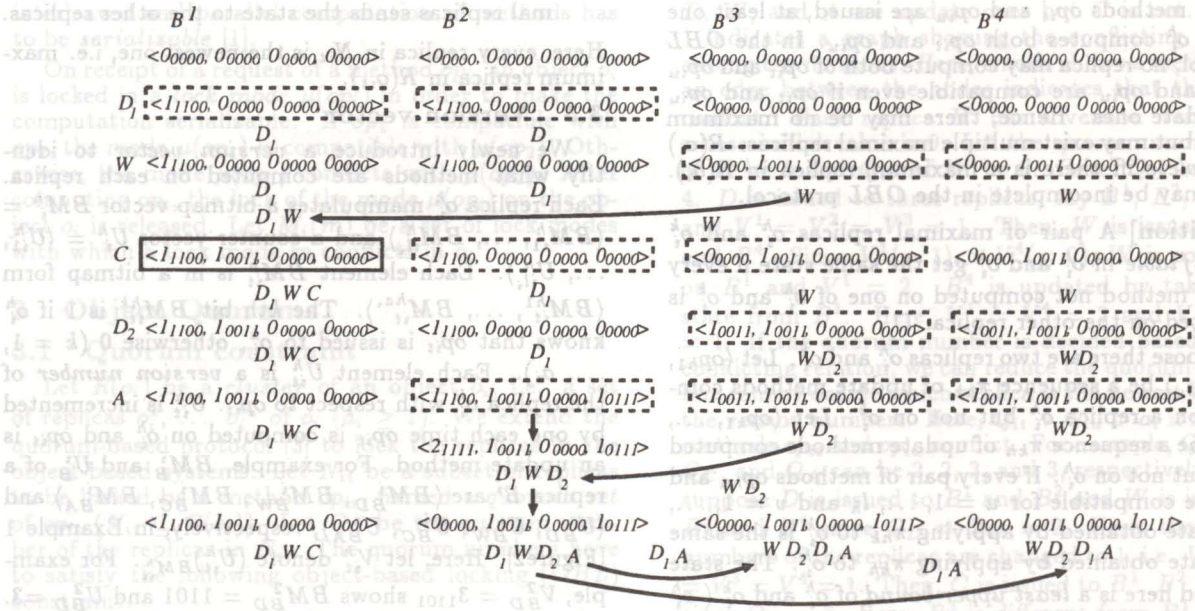

Figure 2: Version vectors.

updated with the state of B^2. $V_B^2 = V_B^3 = V_B^4$. Here, $V_{BD}^1 = V_{BD}^2 = V_{BD}^3 = V_{BD}^4$ $(= 2_{1111})$ is initialized to be 0_{0000} since the same instances of D are surely computed on every replica. In stead of sending the states, B^3 and B^4 can compute A. Then, B^2 can send a sequence of the instances computed on B^2 to the other replicas. □

Let BM_i^h and BM_i^k be bitmaps for o_i^h and o_i^k, respectively. BM_i^h is *included* in BM_i^k $(BM_i^h \subseteq BM_i^k)$ iff $BM_i^{kj} = 1$ if $BM_i^{hj} = 1$ for $j = 1, \ldots, a_i$. $BM_i^h \cup BM_i^k$ shows $\langle BM^1, \ldots, BM^{a_i} \rangle$ where $BM^j = 1$ if $BM_i^{hj} = BM_i^{kj} = 1$, otherwise 0 for $j = 1, \ldots, a_i$.

- $V_{it}^h \leq V_{it}^k$ iff $U_{it}^h \leq U_{it}^k$ and $BM_{it}^h \subseteq BM_{it}^k$.
- $V_i^h \leq V_i^k$ iff $V_{it}^h \leq V_{it}^k$ for every method op_{it}.

A pair of version elements V_{it}^h and V_{it}^k are *not comparable* iff neither $V_{it}^h \leq V_{it}^k$ nor $V_{it}^h \geq V_{it}^k$. If $BM_{it}^h \cap BM_{it}^k \neq \phi$, V_{it}^h and V_{it}^k are not comparable even if $U_{it}^h \leq U_{it}^k$ or $U_{it}^k \leq U_{it}^h$. For example, suppose $V_B^1 = \langle 1_{1100}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$ and $V_B^3 = \langle 2_{1110}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$. Here, $V_B^1 \leq V_B^3$. Here, if $V_B^2 = \langle 2_{0011}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$, V_{BD}^1 and V_{BD}^2 are not comparable. In a subset $N \subseteq R(o_i)$, V_i^h is *maximal* iff there is no replica o_i^k in N where $V_i^k \geq V_i^h$ for o_i^h in N.

We define a least upper bound "$\cup$" for a pair of version elements V_{it}^h and V_{it}^k on op_{it} as follows :

- $V_{it}^h \cup V_{it}^k = \begin{cases} V_{it}^k & \text{if } V_{it}^h \leq V_{it}^k. \\ V_{it}^h & \text{if } V_{it}^h \geq V_{it}^k. \\ \langle U_{it}^h + U_{it}^k,\ BM_{it}^h \cup BM_{it}^k \rangle & \text{otherwise.} \end{cases}$
- $V_i^h \cup V_i^k = \langle V_{i1}^h \cup V_{i1}^k, \ldots, V_{il_i}^h \cup V_{il_i}^k \rangle$.

For example, $V_B^2 = \langle 1_{0011}, 0_{0000}, 0_{0000}, 0_{0000} \rangle \cup V_B^3 = \langle 1_{1100}, 1_{0011}, 0_{0000}, 0_{0000} \rangle = \langle 2_{1111}, 1_{0011}, 0_{0000}, 0_{0000} \rangle$.

[Definition] A version vector V_i^h is *equivalent* with another one V_i^k $(V_i^h \equiv V_i^k)$ iff $V_{iv}^h = V_{iv}^k$ for every update method op_{iv} conflicting with every pair of compatible methods op_{it} and op_{iu}. □

For example, D and W are compatible with one another and conflict with A. Hence, $V_B^1 = \langle 1_{1100}, 0_{0000}, 0_{0000}, 2_{0101} \rangle$ is equivalent with $V_B^2 = \langle 0_{0000}, 1_{0011}, 0_{0000}, 2_{0101} \rangle$ since $V_{BA}^1 = V_{BA}^2 = 2_{0101}$. If $V_i^h \equiv V_i^k$, o_i^h and o_i^k can get the same state by computing compatible methods which are not yet computed on the replicas.

4 Object-based Locking Protocol

We discuss a locking protocol by using the version vector. Suppose a method op_{it} is issued to an object o_i. A replica o_i^h of o_i has a *method log* l_i^h for storing a sequence of method instances computed on o_i^h. Let l_{it}^h be a subsequence of instances of op_{it} in the log l_i^h. Each instance op has a bitmap $op.BM$ showing replicas to which op is issued. That is, $op.BM^k = 1$ if op is issued to o_i^k. The counter U_{it}^h gives the number of method instances in l_{it}^h.

[Locking protocol] An object o_s sends a method op_{it} to every replica in the quorum set N_{it} of op_{it}.

1 All the replicas in N_{it} are locked in a mode $\mu(op_{it})$. Unless succeeded in locking the replicas, op_{it} aborts. Each replica o_i^h in N_{it} sends back

a response with the version vector V_i^h and the method log l_i^h to o_s.

2 On receipt of the responses from all the replicas in N_{it}, o_s obtains $V_s = \cup \{ V_i^k \mid o_i^k \in N_{it} \}$. Let $P_h(op_{it})$ denote a set $\{ op_{iu} \mid op_{iu}$ conflicts with op_{it}, $op_{iu}.BM \neq \langle 1 \ldots 1 \rangle$, and o_i^h computes $op_{iu} \}$. o_s finds a replica o_i^h in N_{it} which is maximal with respect to methods conflicting with op_{it}.

3 If a replica o_i^h is found, o_s requires o_i^h to compute op_{it}. o_i^h computes op_{it}.

- a If op_{it} is not an update method, o_i^h sends a response to o_s.
- b Otherwise, o_i^h sends the set $P_h(op_{it})$ to one replica o_i^k in N_{it}. o_i^k computes every op_{iu} in $P_h(op_{it})$ unless o_i^k had computed op_{iu}. For every op_{iv} in $P_h(op_{it})$, $op_{iv}.BM := op_{iv}.BM \vee op_{it}.BM$. o_i^k sends a response back to o_s.

4 Unless o_i^h is found, o_s selects one maximal replica o_i^h in the quorum set N_{it}. Let $P(op_{it})$ be a set $\{ op_{iu} \mid op_{iu}$ conflicts with op_{it} and is computed on some replica o_i^h in $N_{it} \}$.

- a If op_{it} is an update, o_s sends $P(op_{it})$ to every replica in N_{it}. Each replica o_i^h computes every update op_{iu} computed in N_{it} which is not computed on o_i^h and then computes op_{it}. For every op_{iu} in $P(op_{it})$, $op_{iu}.BM := op_{iu}.BM \vee op.BM$ in o_i^h. o_i^h sends a response back to o_s.
- b If op_{it} is not an update, o_s selects one maximal replica o_i^h in N_{it}. o_s sends $P(op_{it})$ to o_i^h. o_i^h computes every op_{iu} in $P(op_{it})$ if o_i^h had not computed op_{iu}. Then, o_i^h computes op_{it}. o_i^h sends a response to o_s. □

Methods stored in the method log l_i^h have to be eventually removed to reduce the log size in o_i^h. The bitmap $op_{it}.BM$ attached to op_{it} in the log l_i^h shows that o_i^h knows that op_{it} is computed on o_i^k if $BM^k = 1$. If $op_{it}.BM = \langle 1 \ldots 1 \rangle$, o_i^h knows that op_{it} is computed on every replica. However, o_i^h cannot remove op_{it} from l_i^h because another replica o_i^k may not yet know that every replica has computed op_{it}. Hence, a following instance op_{iu} in the log l_i^h is removed :

- $op_{iu}.BM = op_{iv}.BM = \langle 1 \ldots 1 \rangle$ for every method op_{iv} in l_i^h which conflicts with op_{iu} and is computed before op_{iu}.

The counter U_{it}^h and the bitmap BM_{it}^h are initialized again, i.e. $U_{it}^h := 0$ and $BM_{it}^h := \langle 0 \ldots 0 \rangle$ if BM_{it}^h gets $\langle 1 \ldots 1 \rangle$. If $BM_{it}^h = \langle 1 \ldots 1 \rangle$ in some replica o_i^h, op_{it} is computed on every replica o_i^k in the quorum set N_{it}. Thus, U_{it}^h shows how many instances of op_{it} are computed on o_i^t. If $BM_{it}^h \cap BM_{it}^k = \phi$ and ($U_{it}^h > 0$ or $U_{it}^k > 0$) for an update method op_{it}, a sequence s^h of instances of op_{it} computed on o_i^h is different from s^k of o_i^k. s^h and s^k include U_{it}^h and U_{it}^k instances of op_{it}, respectively. In the OBL protocol, s^k and s^h are required to be computed on o_i^h and o_i^k, respectively. Then, $BM_{it}^h := BM_{it}^k := BM_{it}^h \cap BM_{it}^k$ and $U_{it}^h := U_{it}^k := U_{it}^h + U_{it}^k + 1$.

[Theorem] For every update method op_{it}, if $BM_{it}^h \subseteq BM_{it}^k$ and $U_{it}^h \leq U_{it}^k$, every instance of op_{it} computed on B_i^h is also computed on B_i^k. □

[Theorem] If $V_i^h \leq V_i^k$, every pair of conflicting instances op_{it} and op_{iu} computed on a replica o_i^h are computed on another replica o_i^k in the same order. □

[Theorem] Every pair of maximal replicas o_i^h and o_i^k are unifiable to one unique replica. □

[Theorem] All the replicas are unifiable to one unique replica in the OBL protocol. □

5 Evaluation

We evaluate the OBL protocol by comparing with the quorum-based protocol in terms of the number of the replicas locked. An object o_i supports methods $op_{i1}, \ldots, op_{il_i}$ ($l_i \geq 1$). In the quorum-based protocol, an update op_{it} is considered to be *write*. Otherwise, op_{it} is *read*. Quorum sets N_{it} and N_{iu} for a pair of methods op_{it} and op_{iu} are decided so that $N_{it} \cap N_{it} \neq \phi$ if op_{it} or op_{iu} is an update. On the other hand, N_{i1}, $\ldots$, N_{il_i} are obtained based on the conflicting relation. $N_{it} \cap N_{iu} \neq \phi$ only if op_{it} conflicts with op_{iu}. That is, $N_{it} \cap N_{iu} = \phi$ if op_{it} is compatible with op_{iu} even if op_{it} or op_{iu} is an update one in the OBL protocol. Let $\varphi(op_{it})$ denote a usage frequency of op_{it} where $\varphi(op_{i1}) + \ldots + \varphi(op_{il_i}) = 1$. For various values of the usage frequency φ, the minimum quorum numbers Q_{it}^O and Q_{it}^Q to be locked in the OBL protocol and the quorum-based protocol, respectively, are calculated. In the evaluation, an object is assumed to support two types of methods op_1 and op_2. Figure 3 shows conflicting graphs for three cases.

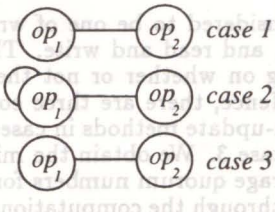

Figure 3: Conflicting relations.

Figures 4, 5, and 6 show the quorum number for usage frequency of op_1 in case that the number a_i of the replicas is 10. In the quorum-based protocol, the quorum number of op_i depends on whether or not op_i is an update. We assume that op_i is an update if op_i conflicts with itself. We also assume that one of op_1 and op_2 is an update if op_1 conflicts with op_2. For example, at least one of op_1 and op_2 must be an update in case 1 because op_1 conflicts with op_2. Hence, op_1

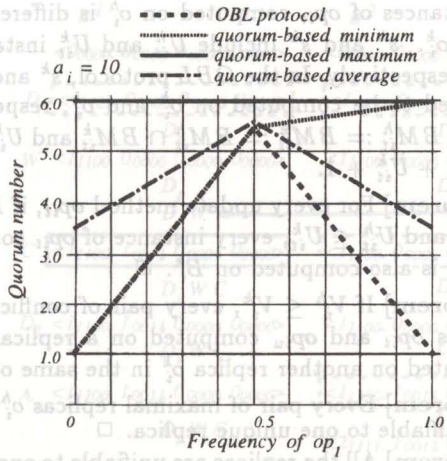

Figure 4: Evaluation of OBL protocol (case 1).

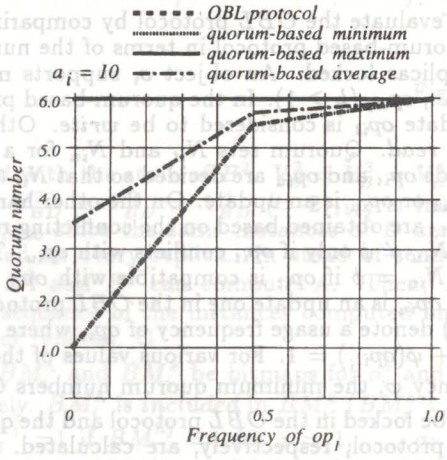

Figure 5: Evaluation of OBL protocol (case 2).

and op_2 are considered to be one of write and write, write and read, and read and write. There are three cases depending on whether or not the methods are update ones. Hence, there are three combinations of update and non-update methods in case 1, two in case 2, and four in case 3. We obtain the minimum, maximum, and average quorum numbers for the quorum-based protocol through the computation. The quorum number of the *OBL* protocol is also calculated so that the *OBL* constraint are satisfied. Figures 4, 5, and 6 show that the fewer number of replicas are locked in the *OBL* protocol than the quorum-based protocol. Even if op_1 and op_2 are update methods, op_1 may be compatible with op_2. For example, the method D is compatible with W in Example 1.

6 Concluding Remarks

This paper has discussed the *object-based locking* (*OBL*) protocol for the replicas of the objects. The

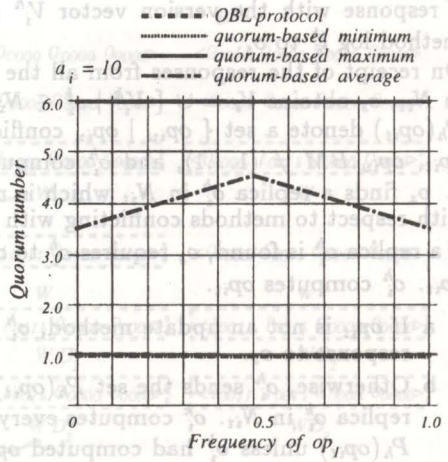

Figure 6: Evaluation of OBL protocol (case 3).

object supports a more abstract level of method than *read* and *write*. The version vector has been proposed to maintain the mutual consistency of the replicas. The replicas are not required to compute every update instance which has been computed on the other replicas if the instance is compatible with the instances computed. Through the evaluation, we have shown that fewer number of replicas are locked in the *OBL* protocol than the quorum-based protocol.

References

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

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

[3] Garcia-Molina, H. and Barbara, D., "How to Assign Votes in a Distributed System," *Journal of ACM*, Vol 32, No.4, 1985, pp. 841-860.

[4] Hasegawa, K. and Takizawa, M., "Optimistic Concurrency Control for Replicated Objects", *Proc. of the Int'l Symp. on Communications (ISCOM'97)*, 1997, pp. 149–152.

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

[6] Korth, H. F., "Locking Primitives in a Database System," *JACM*, Vol. 30, No. 1, 1983, pp. 55-79.

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

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