Vol. 40 No. 1 Transactions of Information Processing Society of Japan Jan. 1999

Regular Paper

Version Vector for Maintaining Distributed Replicas

KYOUJI HASEGAWA,† HIROAKI HIGAKI,† and MAKOTO TAKIZAWA†

In object-based systems, objects are replicated to increase the performance, reliability, and availability. We discuss a novel object-based locking (OBL) protocol for locking replicas of objects that support abstract methods; this protocol was created by extending the quorum-based protocol to abstract objects. Here, the number of the replicas locked is decided on the basis of the frequency of use of the lock mode. 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 but on another one are computed when a method conflicting with them is issued to the replica. Hence, the replicas have the same state. We propose a version vector for identifying what methods are computed on a replica.

1. Introduction

To increase the reliability, availability, and performance of an object-based system, objects are replicated in the system. Replicas of an 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*. This method 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. Q_r and Q_w are decided on the basis of how frequently *read* and *write* are issued.

Distributed applications are modeled as a collection of multiple cooperating objects. Object-based frameworks such as CORBA [8)] are widely used for developing distributed applications. Objects support abstract methods such as *Deposit* in a *bank* object. Each 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 the *OBL* (*object-based locking*) protocol. The *OBL* protocol is an extension of the quorum-based protocol [3)] to abstract objects. Before computing a method op_t on an object o, some number Q_t of the replicas of o are locked in the abstract mode for op_t. Q_t is the *quorum number* of op_t. Q_t depends on how *frequently* op_t is invoked. The more frequently op_t is invoked, the fewer replicas are locked. Suppose a pair of methods op_t and op_u are issued to replicas of the object o. If op_t and op_u change the state of o, the traditional quorum-based protocol requires that "$Q_t + Q_u > a$" hold. That is, both op_t and op_u are guaranteed to be computed on at least one replica. If op_t and op_u are computed on replicas o^t and o^u, respectively, the states of o^t and o^u are different. Then, if both op_t and op_u are computed on replicas o^t and o^u, respectively, o^t and o^u have the same state if op_t and op_u are compatible. In the quorum-based method, there must be at least one newest replica where every write method is computed. However, it is possible in some instances to construct the newest version even if there is no up-to-date replica. To do so, it is necessary to identify what methods are computed on each replica. We propose a *version vector* to identify the methods computed. In the *OBL* protocol, fewer number of replicas are locked than in the quorum-based protocol and the *2PL* protocol.

In Section 2, we present the system model. In Section 3, we discuss abstract lock modes. In Section 4, we discuss the *OBL* protocol with the version vector. In Section 5, we evaluate the *OBL* protocol in comparison with the quorum-based protocol in terms of the number of replicas to be locked.

2. Replicated Objects

A system is composed of objects $o_1, \ldots, o_n$ ($n \geq 1$) that cooperate by exchanging messages in a network. Each object o_i supports a set of

† Department of Computers and Systems Engineering, Tokyo Denki University

methods $op_{i1}, \ldots, op_{il_i}$, and is encapsulated so that it can be manipulated only through the methods, it supports. By using the network, o_i can send messages to o_j with no message loss, in the order in which they were sent.

If a transaction T sends a request message for a method op_i to an object o_i, o_i computes the method op_i, which may invoke a method op_{ij} on another object o_{ij}. o_i sends op_i back to T as a response. The transaction T is an atomic invocation sequence of methods. T *commits* only if all the methods invoked by T successfully complete their tasks, that is, if the methods commit. A method op invoked by T commits only if all the methods invoked by op commit. op is also an atomic unit of computation. Thus, the methods are *nested*[7]. A method that changes the state of the object is an *update* method.

Let $op(s)$ denote a state obtained by applying a method op to a state s of o_i. A method 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_i. op_{ij} *conflicts* with op_{ik} unless op_{ij} is compatible with op_{ik}. The conflicting relation is not transitive. We assume that the conflicting relation is symmetric. The interleaved and parallel computation of methods has to be *serializable*[1].

On receipt of a request from 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 a method op_1 is compatible with op_2, $\mu(op_1)$ is compatible with $\mu(op_2)$. Otherwise, the modes conflict. For example, *Dlock* and *Wlock* denote lock modes of the methods *Deposit* and *Withdraw* of the bank object B, respectively. *Dlock* and *Wlock* are compatible. After computing op_i, the lock of the mode $\mu(op_i)$ on o_i is released. Let $C_i(m)$ be a set of lock modes with which a lock mode m conflicts in o_i.

3. Abstract Lock Modes

Replicas of an object o_i are locked in a mode $\mu(op_i)$ before op_i is computed. If the replicas are already locked by modes conflicting with $\mu(op_i)$, op_i blocks. The larger number of modes conflict with $\mu(op_i)$, the longer op_i blocks. A lock mode m_1 is *more restricted* than m_2 iff $C_i(m_1) \supseteq C_i(m_2)$[6].

[Example 1] Let *rlock* and *wlock* be lock modes for *read* and *write*, respectively, in a file object f. $C_f(rlock) = \{wlock\}$ and $C_f(wlock) = \{wlock, rlock\}$. Since $C_f(rlock) \subseteq C_f(wlock)$, *wlock* is more restricted than *rlock*. □

[Example 2] A *bank* object B supports methods *Deposit*, *Withdraw*, *Check*, and *Audit*. Let *Dlock*, *Wlock*, *Clock*, and *Alock* show the lock modes $\mu(Deposit)$, $\mu(Withdraw)$, $\mu(Check)$, and $\mu(Audit)$, respectively. *Dlock* and *Wlock* are compatible. The mode *Clock* conflicts with the modes *Dlock* and *Wlock*. *Audit* derives data on the accounts and stores them in the object B. Hence, *Alock* conflicts with *Dlock* and *Wlock* but is compatible with *Clock*. $C_B(Dlock) = C_B(Wlock) = \{Alock, Clock\}$ and $C_B(Clock) = C_B(Alock) = \{Dlock, Wlock\}$. Since $C_B(Dlock) \cap C_B(Clock) = \phi$, there is no relation among *Clock* and *Dlock*. □

Let $\varphi(m)$ be the frequency of use of a mode m, that is, the frequency with which methods of m are issued to o_i. Here, $\sum_{m \in M_i} \varphi(m) = 1$. The *weighted strength* $||C_i(m)||$ of m is defined to be $\sum_{m' \in C_i(m)} \varphi(m')$ (≤ 1).

[Definition] A mode m_1 is *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 $||C_i(m_1)|| \geq ||C_i(m_2)||$. □

Let m_1 and m_2 be modes $\mu(op_1)$ and $\mu(op_2)$ in an object o_i, respectively. If $||C_i(m_1)|| \geq ||C_i(m_2)||$, there is a bigger blocking probability that op_1 will wait for the release of the lock conflicting with op_1 than op_2.

In Example 1, $wlock \succeq rlock$ since $C_f(wlock) \supset C_f(rlock)$. In Example 2, suppose the usage ratios are $\varphi(Alock) = 0.1$, $\varphi(Clock) = 0.2$, $\varphi(Dlock) = 0.4$, and $\varphi(Wlock) = 0.3$. $||C_B(Clock)|| = ||C_B(Alock)|| = \varphi(Dlock) + \varphi(Wlock) = 0.7$. $||C_B(Dlock)|| = ||C_B(Wlock)|| = \varphi(Clock) + \varphi(Alock) = 0.3$. Hence, $Dlock \preceq Clock$, $Wlock \preceq Clock$, and $Alock \preceq Clock$, since $||C_B(Dlock)|| < ||C_B(Clock)||$, $||C_B(Wlock)|| < ||C_B(Dlock)||$, and $||C_B(Alock)|| \leq ||C_B(Clock)||$.

The modes supported by the object o_i are partially ordered according to the strength relation "$\preceq$". A mode m is referred to as *maximal* in o_i iff there is no mode m' such that $m \preceq m'$. The *minimal* mode is similarly defined. m is the *least upper bound* of modes m_1 and m_2 (written as $m_1 \cup m_2$) iff $m_1 \preceq m$, $m_2 \preceq m$, and there is no mode m' such that $m_1 \preceq m' \preceq m$ and $m_2 \preceq m' \preceq m$. The *greatest lower bound* of m_1 and m_2 (written as $m_1 \cap m_2$) is similarly defined.

We discuss how to lock replicas $o_i^1, \ldots, o_i^{a_i}$ in the cluster $R(o_i)$. We extend the traditional quorum-based protocol[3] to the replicas

in object-based systems. Let N_{it} be a set of replicas to be locked by a method op_{it}, named the *quorum set* of op_{it}. Let Q_{it} be the number of the replicas in N_{it}, named the *quorum number* of op_{it}. The larger the number of replicas locked, the more communication and computation are required. Hence, the more frequently op_{it} is invoked, the fewer replicas are locked. The quorum number Q_{it} is decided so as to satisfy the following constraints:

[OBL constraints]

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

In addition, the quorum numbers $Q_{i1}, \ldots, Q_{il_i}$ are obtained so as to minimize the average number $\varphi(op_{i1})Q_{i1} + \ldots + \varphi(op_{il_i})Q_{il_i}$ of replicas locked. A transaction T locks replicas $o_i^1, \ldots, o_i^{a_i}$ of o_i by the following locking protocol before manipulating the replicas by a method op_{it}.

[Locking protocol]

1 First, a quorum set N_{it} is fixed for op_{it} in the cluster $R(o_i)$.
2 Every replica in N_{it} is locked in a mode $\mu(op_{it})$.
3 If all the replicas in N_{it} are locked, the replicas in N_{it} are manipulated by op_{it}.
4 When T commits, the locks on the replicas in N_{it} are released. □

4. Object-based Locking Protocol

4.1 Quorums for Methods

In the quorum-based protocol, $Q_{it}+Q_{iu} > a_i$ if a pair of methods op_{it} and op_{iu} are update methods where a_i is the number of 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 other words, $Q_{it}+Q_{iu} > a_i$ might 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 one replica computes both 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. □

[Example 3] Replicas B^1, B^2, B^3, and B^4 of the bank object B support the four methods *Deposit* (D), *Withdraw* (W), *Check* (C), and *Audit* (A) shown in Example 2. D is compatible with W. C is compatible with A. D and W conflict with C and A. D, W, and A are update methods 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 two methods shown by 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 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 sending the state of B^1. Here, $V^1 = V^4 = 2$ and $V^2 = V^3 = 1$. If the quorum number is decided on the basis of the conflicting relation among the methods, we can reduce the quorum number but cannot decide which replica is up-to-date by using the version numbers. Here, $Q_{it} + Q_{iu} > a_i$ if the 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 that the method D is issued to two replicas 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, that is, $V^1 = V^2 = V^3 = V^4 = 1$. Then, C is issued to B^1, B^2, and B^3 since $Q_C = 3$. Here, the state of B^3 is different from the states of 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 the replicas 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 brought up-to-date by computing methods that are not computed on the replicas but computed on the others if these methods are compatible. The problem is that we cannot tell that the states of B^1 and B^2 are different from those of B^3 and B^4 by locking at the version numbers of the replicas, because the version numbers of B^1, B^2, B^3, and B^4 are all 1 after D and W have been computed. □

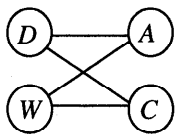


Fig. 1 Conflicting graph.

A replica o_i^h is considered to be *newer* than another replica o_i^k if every method computed in o_i^k is computed on o_i^h. We define a precedent relation "$\rightarrow$" among the replicas to show which

replica is newer.

[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 a replica o_i^k is obsolete, since some update methods computed on o_i^h are not computed on o_i^k. The quorum-based protocol requires that every quorum set include at least one maximum replica o_i^h, that is, $o_i^h \rightarrow o_i^k$ for every replica o_i^k. In other words, if a pair of update methods op_{it} and op_{iu} are issued, at least one replica o_i^h computes both of them. On the other hand, in the *OBL* protocol, no replica may compute both op_{it} and op_{iu} if they are compatible, even if they are update methods. Hence, there may be no maximum replica but multiple maximal replicas. A cluster $R(o_i)$ is *complete* iff there is a maximum replica in $R(o_i)$. Although $R(o_i)$ is complete in the quorum-based method, $R(o_i)$ may be incomplete in the *OBL* protocol.

[Definition] Every pair of maximal replicas o_i^h and o_i^k are *unifiable* ($o_i^h \equiv o_i^k$) iff they have the same state in the event that every update method not computed on one of o_i^h and o_i^k is computed on the other replica. □

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 referred to as the *least upper bound* of o_i^h and o_i^k (written as $o_i^h \cup o_i^k$) on to the precedent relation "$\rightarrow$". For example, suppose the replicas B^1 and B^2 compute the method D and the replicas B^3 and B^4 compute W in Example 3. 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 $\{o_i^k \mid o_i^k \equiv o_i^h \text{ in } R(o_i)\}$ for a maximal replica o_i^h. A cluster $R(o_i)$ is *consistent* iff $U(o_i^h) = U(o_i^k)$ for every pair of maximal replicas o_i^h and o_i^k in $R(o_i)$. Here, $U(o_i^h)$ is referred to as a *unifiable* set $U(o_i)$ of $R(o_i)$. A least upper bound 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.

In an incomplete cluster $R(o_i)$, some methods computed on a maximal replica have to be computed later on other maximal replicas that have not yet computed the methods. *Incomplete methods* are defined as update methods that are computed on some replicas but not on every replica in a unifiable set $U(o_i)$ of $R(o_i)$. In Example 3, the methods D and W are incomplete. The replicas B^1 and B^2 are unifiable, i.e., $B^1 \equiv B^2$. $U(o_i) = \{B^1, B^2, B^3, B^4\}$. Every pair of incomplete methods not computed on the same replica are not computed on any replica. Complete methods are update methods computed on every replica in $U(o_i)$.

Let us consider how a method 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 that 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, the method op_{it} is computed as follows:

1 Incomplete methods on each maximal replica are computed on the other maximal replicas in the quorum set N_{it}, as explained 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 in the maximal ones but not computed in 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, that is, the maximum replica in $R(o_{it})$.

4.2 Version Vector

In the *OBL* protocol, it is critical to identify what methods each replica has computed. As explained before, the version number cannot be used to maintain consistency among the object replicas, because some pairs of update methods may not be computed on any replica if they do not conflict.

We introduce a *version vector* to identify what methods are computed on each replica. Each replica o_i^h has a version 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 a version of the replica o_i^h for a method op_{it} in a bitmap form $\langle BM_{it}^{h1}, \ldots, BM_{it}^{ha_i} \rangle$ showing to which replica op_{it} is issued. The kth bit BM_{it}^{hk} is 1 if op_{it} is issued to o_i^k; otherwise, 0 ($k = 1, \ldots, a_i$). Each U_{it}^h is a *version number* of o_i^h

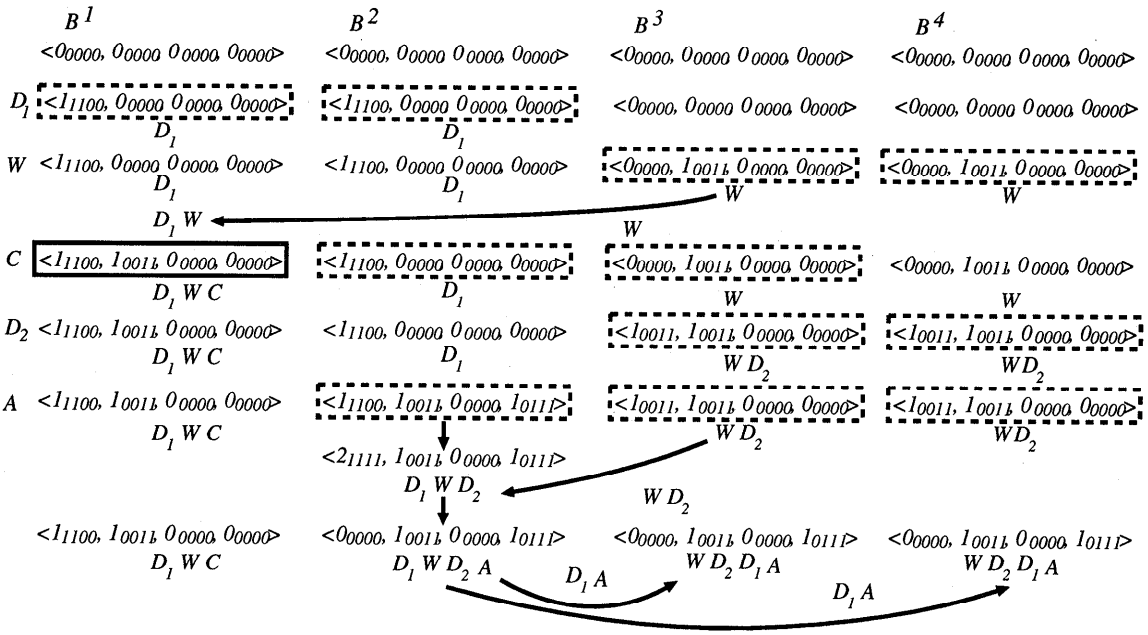


Fig. 2 Version vectors.

with respect to op_{it}. U_{it}^h is incremented by 1 each time op_{it} is computed on o_i^h and op_{it} is an update method. For example, vectors 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 3 (**Fig. 2**). Here, let V_{it}^h denote $U_{it\,BM_{it}^h}^h$. For example, $V_{BD}^2 = 2_{1111}$ shows $BM_{BD}^2 = 1111$ and $U_{BD}^2 = 2$. This means that two method instances D_1 and D_2 of D are computed on the replica B^2, and B^2 knows that the instances are also computed on $B^1, \ldots B^4$. V_i^h is a version vector $\langle V_{i1}^h, \ldots, V_{il_i}^h\rangle$. Initially, $V_B^j = \langle 0_{0000}, 0_{0000}, 0_{0000}, 0_{0000}\rangle$ in each replica B^j for $j = 1, \ldots, 4$. Suppose D is issued to B^1 and B^2. $V_B^1 = V_B^2 = \langle 1_{1100}, 0_{0000}, 0_{0000}, 0_{0000}\rangle$. Then, the method W is issued to B^3 and B^4. $V_B^3 = V_B^4 = \langle 0_{0000}, 1_{0011}, 0_{0000}, 0_{0000}\rangle$. Suppose the method C is issued to B^1, B^2, and B^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 method, V_{BC}^1 is not changed. Suppose 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 the replicas B^2, B^3, and B^4. Since $V_{BD}^2 = 1_{1100}$ and $V_{BD}^3 = 1_{0011}$, B^2 and B^3 exchange the instances of D computed with one another. In addition, $V_{BW}^2 = 0_{0000}$ and $V_{BW}^3 = 1_{0011}$. Here, the instances of the methods 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 and W are compatible, 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. $V_B^2 = \langle 2_{1111}, 1_{0011}, 0_{0000}, 1_{0111}\rangle$ and A changes the state of B^2. If the state of B^2 is sent to B^3 and B^4, B^3 and B^4 are 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 certain to be computed on every replica. Instead of sending the states to B^3 and B^4, B^3 and B^4 can compute A. Then, B^2 can send a sequence of the method instances computed on B^2 to the other replicas.

Let BM_i^h and BM_i^k be bitmaps for the replicas 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 $BM_i^{hj} = BM_i^{kj} = 1$ and $BM^j = 0$ otherwise 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}.

If $BM_{it}^h \cap BM_{it}^k \neq \phi$, V_{it}^h and V_{it}^k are not ordered 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_B^1 and V_B^2 are not compared. 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 operation $\cup$ for a pair of vector 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, suppose $V_B^2 = \langle 1_{0011}, 0_{0000}, 0_{0000}, 0_{0000}\rangle$ and $V_B^3 = \langle 1_{1100}, 1_{0011}, 0_{0000}, 0_{0000}\rangle$. $V_B^2 \cup V_B^3 = \langle 2_{1111}, 1_{0011}, 0_{0000}, 0_{0000}\rangle$.

[Definition] A version vector V_i^h is *equivalent* to 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. □

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

4.3 Locking Protocol

We discuss a locking protocol with the version vector. Suppose a method op_{it} is issued to an object o_i. A replica o_i^h 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 method log l_i^h. Here, let $op.BM$ be a bitmap showing replicas to which an instance op is issued. That is, $op.BM^k = 1$ if op is issued to a replica 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 the quorum set N_{it} are locked in a mode $\mu(op_{it})$. Unless the replicas are successfully locked, the method op_{it} aborts. Each replica o_i^h in N_{it} sends 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 the quorum set 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} that is maximal with respect to a method conflicting with op_{it}.

3 If a replica o_i^h is found at step 2, o_s requires o_i^h to compute the method 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 N_{it}. o_i^k computes every method op_{iu} in $P_h(op_{it})$ unless o_i^k has 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 to o_s.

4 Unless o_i^h is found at step 2, o_s selects one maximal replica o_i^h in 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 method, o_s sends the set $P(op_{it})$ to every replica in N_{it}. Each replica o_i^h computes every update method op_{iu} computed in N_{it} which is not computed on o_i^h and then computes op_{it}. For every method 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 to o_s.

   b If op_{it} is not an update method, 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 has not computed op_{iu}, and then o_i^h computes op_{it}. o_i^h sends a response to o_s. □

Methods stored in the log l_i^h have to be eventually removed in order to reduce the size of the log in the replica o_i^h. The bitmap $op_{it}.BM$ in the log l_i^h shows that o_i^h knows that the instance 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 the log l_i^h, because another replica o_i^k may not yet know that every replica has computed op_{it}. Hence, a method instance op_{iu} in the log l_i^h is removed as follows:

- op_{iu} is removed from the log l_i^h if $op_{iu}.BM = op_{iv}.BM = \langle 1 \ldots 1\rangle$ for every method

op_{iv} in l_i^h that conflicts with op_{iu} and is computed before op_{iu}.

The counter U_{it}^h and the version 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, the method 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 $BM_{il}^h \cap BM_{il}^k = \phi$ 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 a sequence s^k in o_i^k. The sequences s^h and s^k include U_{it}^h and U_{it}^k instances of op_{it}, respectively. In the OBL protocol, the sequences 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 $U_{it}^h = U_{it}^k = U_{it}^h + U_{it}^k + 1$.

[Proposition] For every update method op_{il}, $BM_{it}^h = BM_{it}^k$ and $U_{it}^h = U_{it}^k$ if $BM_{it}^h \cap BM_{it}^k \neq \phi$. □

[Proposition] For every update method op_{it}, $U_{it}^h \leq U_{it}^k$ if $BM_{it}^h \subseteq BM_{it}^k$. □

[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. □

From the properties of the OBL protocol, it is straightforward to show that the following theorem hold:

[Theorem] If $V_i^h \leq V_i^k$, every pair of conflicting method 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 can be unified to create one unique replica. □

[Proof] Assume that a pair of methods op_{it} and op_{iu} conflict. If op_{it} is computed on the replica o_i^h, op_{iu} is also computed on o_i^h. Otherwise, o_i^h is not maximal. Here, both op_{it} and op_{iu} are computed on both o_i^h or o_i^k, or on neither of them, from the OBL properties. Method instances computed on either o_i^h or o_i^k do not conflict. These methods can be computed in any order. Therefore, if the instances computed on one replica are computed on the other replica, o_i^h and o_i^k have the same state. □

From these theorems, the following theorem holds:

[Theorem] All the replicas can be unified to create one unique replica in the OBL protocol. □

5. Evaluation

We evaluate the OBL protocol by comparing it with the traditional quorum-based protocol in terms of the number of replicas locked. An object o_i supports methods $op_{i1}, \ldots, op_{il_i}$ $(l_i \geq 1)$. In the quorum-based protocol, a method op_{it} of an object o_i is considered to be *write* if op_{it} is an update method. 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 method. On the other hand, the quorum sets $N_{i1}, \ldots, N_{il_i}$ are obtained on the basis of the conflicting relation among the methods. $N_{it} \cap N_{iu} \neq \phi$ 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 method in the OBL protocol. Let $\varphi(op_{it})$ denote the frequency of use of a method op_{it}, where $\varphi(op_{i1}) + \ldots + \varphi(op_{il_i}) = 1$. Let C_i denote a conflicting relation, where $C_{itu} = 1$ if op_{it} conflicts with op_{iu}, and $C_{itu} = 0$ otherwise. The minimum quorum numbers Q_{it}^O and Q_{it}^Q of the replicas to be locked in the OBL protocol and the quorum-based protocol, respectively, are calculated for various values of the frequency of use φ and C_i.

In the evaluation, an object o_i is assumed to support two methods op_1 and op_2. **Figure 3** shows conflicting graphs for three cases of the conflicting relation C_i. In the first case, op_1 conflicts with op_2. In the second case, op_1 conflicts with op_1 and op_2. In the third case, op_1 is compatible with op_2.

Figures 4, **5**, and **6** show the quorum number, that is, the number of replicas locked for usage frequency of op_1 in the case where there are ten replicas $(a_i = 10)$. In the traditional protocol, the quorum number of the method op_i depends on whether or not op_i is an update method. We assume that op_i is an update method if op_i conflicts with itself and that either op_1 or op_2 is an update method if op_1

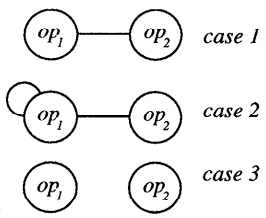


Fig. 3 Conflicting relations.

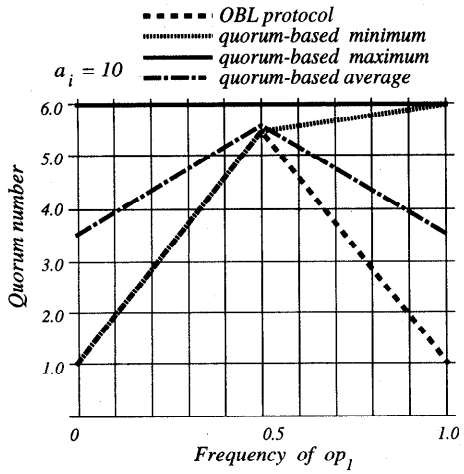


Fig. 4 Evaluation of OBL protocol (case 1).

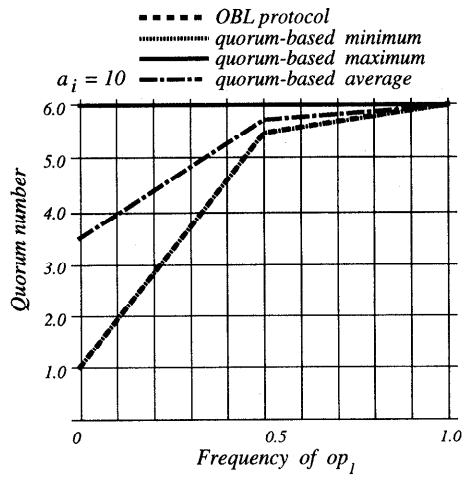


Fig. 5 Evaluation of OBL protocol (case 2).

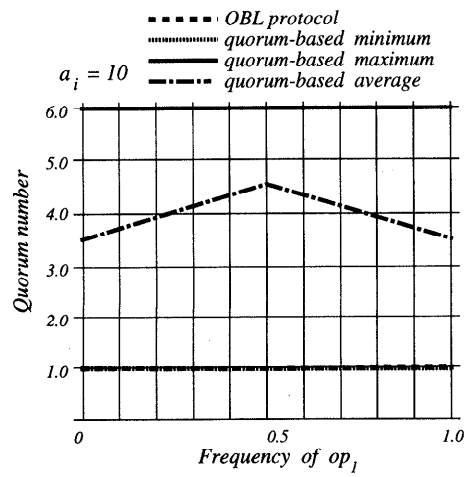


Fig. 6 Evaluation of OBL protocol (case 3).

conflicts with op_2. For example, at least one of op_1 and op_2 must be an update method in case 1, because op_1 conflicts with op_2. Hence, op_1 and op_2 are considered to be one of three pairs: write and write, write and read, or read and write. That is, there are three cases, depending on whether the methods are update ones or not. Hence, there are three combinations of update and non-update methods in case 1, two in case 2, and four in case 3. By computation, we obtain the minimum, maximum, and average quorum numbers for the quorum-based protocol. The quorum number of the *OBL* protocol is also calculated so that the *OBL* constraint presented in Section 4 is satisfied. Figures 4, 5, and 6 show that fewer replicas are locked in the *OBL* protocol than in the quorum-based protocol. In the *OBL* protocol, the conflicting relation among the methods is defined on the basis of the semantics of the object. Even if the methods op_1 and op_2 are update methods, op_1 may be compatible with op_2. For example, method D is compatible with W in Example 3.

6. Concluding Remarks

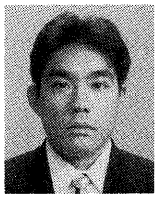
We have discussed an *object-based locking* (*OBL*) protocol for replicas of objects. Objects support more abstract level of methods than *read* and *write*. The strength relation among the lock modes is defined on the basis of the conflicting relation among the methods and the frequencies of use of the methods. In addition, we proposed the version vector for maintaining the mutual consistency of the replicas. The replicas are not required to compute every update method instance that has been computed on the other replicas if the instance is compatible with the instances computed. Our evaluation showed that by using the *OBL* protocol, more efficient access to replicated objects can be realized in a distributed system, since fewer replicas are locked than in the traditional quorum-based protocol.

References

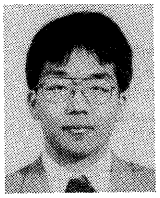
1) Bernstein, P.A., Hadzilacos, V. and Goodman, N.: *Concurrency Control and Recovery in Database Systems*, Addison-Wesley (1987).
2) Carey, J.M. and Livny, M.: Conflict Detection Tradeoffs for Replicated Data, *ACM TODS*, Vol.16, No.4, pp.703–746 (1991).
3) Garcia-Molina, H. and Barbara, D.: How to Assign Votes in a Distributed System, *J. ACM*, Vol.32, No.4, pp.841–860 (1985).

4) Hasegawa, K. and Takizawa, M.: Optimistic Concurrency Control for Replicated Objects, *Proc. Int'l Symp. on Communications (ISCOM '97)*, pp.149–152 (1997).
5) Jing, J., Bukhres, O. and Elmagarmid, A.: Distributed Lock Management for Mobile Transactions, *Proc. IEEE ICDCS-15*, pp.118–125 (1995).
6) Korth, H.F.: Locking Primitives in a Database System, *JACM*, Vol.30, No.1, pp.55–79 (1983).
7) Moss, J.E.: Nested Transactions: An Approach to Reliable Distributed Computing, *The MIT Press Series on Information Systems* (1985).
8) Silvano, M. and Douglas, C.S.: Constructing Reliable Distributed Communication Systems with CORBA, *IEEE Communications Magazine*, Vol.35, No.2, pp.56–60 (1997).
9) Yoshida, T. and Takizawa, M.: Model of Mobile Objects, *Proc. 7th DEXA*, Lecture Notes in Computer Science, Vol.1134, pp.623–632, Springer-Verlag (1996).

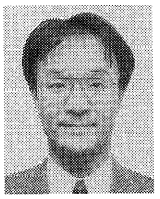
(Received May 8, 1998)
(Accepted November 9, 1998)



Kyouji Hasegawa was born in 1974. He received his B.E. degree in computers and systems engineering from Tokyo Denki University, Japan in 1997. He is now a graduate student of the master course in the Dept. of Computers and Systems Engineering, Tokyo Denki Univ. His research interests include distributed database system and object management systems.



Hiroaki Higaki was born in Tokyo, Japan, in 1967. He received the B.E. degree from the University of Tokyo in 1990. He received the D.E. degree in 1997. His research interests include distributed algorithms and computer network protocols. He is a member of IEEE CS, ACM and IEICE.



Makoto Takizawa was born in 1950. He received his B.E. and M.E. degrees in Applied Physics from Tohoku University, Japan, in 1973 and 1975, respectively. He received his D.E. in Computer Science from Tohoku Univ. in 1983. From 1975 to 1986, he worked for Japan Information Processing Developing Center (JIPDEC) supported by the MITI. He is currently a Professor of the Dept. of Computers and Systems Engineering, Tokyo Denki Univ. since 1986. From 1989 to 1990, he was a visiting professor of the GMD-IPSI, Germany. He is also a regular visiting professor of Keele Univ., England since 1990. He was a program co-char of IEEE ICDCS-18, 1998 and serves on the program committees of many international conferences. His research interests include communication protocols, group communication, distributed database systems, transaction management, and security. He is a member of IEEE, ACM, IPSJ, and IEICE.