

Quorum-Based Replication in Object-Based Systems

KATSUYA TANAKA, KYOJI HASEGAWA AND MAKOTO TAKIZAWA

Department of Computers and Systems Engineering

Tokyo Denki University

Hiki-gun, Saitama 350-0394, Japan

E-mail: {katsu, kyo, taki}@takilab.k.dendai.ac.jp

Objects are replicated in order to realize object-based systems. We discuss a novel quorum-based object locking (QOL) protocol which can be used to lock replicated objects by extending the traditional quorum-based protocols for simple read and write operations to abstract methods. If a pair of methods op_1 and op_2 are compatible, the summation of the quorum numbers of op_1 and op_2 can be smaller than the number of replicas in the QOL protocol even if op_1 or op_2 change the state of the object. We newly propose a version vector which can be used to identify which methods are performed on a replica. We discuss the QOL protocol, where the replicas exchange the compatible methods performed by themselves but not by others, by means of the version vector.

Keywords: quorum-based object locking (QOL) protocol, replication of objects, concurrency control, object-based system, object vector

1. INTRODUCTION

Distributed applications are composed of objects in object-based frameworks like CORBA [14]. In order to increase reliability, availability, and performance, the objects are replicated in the system. The replicas of an object have to be mutually consistent. In the two-phase locking (2PL) protocol [2, 4], one of the replicas for *read* and all the replicas for *write* are locked according to the *read-one-write-all* scheme. The 2PL protocol is not efficient for write-dominated applications because all the replicas are locked for *write*. In the quorum-based protocol [7], some numbers N_r and N_w of the replicas called *quorum numbers*, are locked for *read* and *write* methods, respectively, where " $N_r + N_w > a$ " in which the number of replicas is a . A subset of replicas is referred to as a *quorum*. The quorum numbers N_r and N_w can be decided so that the more write requests, there are, the smaller N_w is. The papers in [1, 6] discuss how to decide on the quorum numbers N_r and N_w .

An object only supports procedures, called *methods*, by means of which the object can be manipulated. The object is locked in an abstract mode for a method, e.g. *Deposit lock* mode for a *Bank* object. A pair of methods op_i and op_u supported by an object o conflict if the result obtained by performing op_i and op_u depends on the computation order of op_i and op_u [2]. If the object o is locked in a mode conflicting with a method op , then op blocks.

In this paper, we propose a novel *QOL* (*quorum-based object locking*) protocol for replicated objects, which extends the quorum-based protocol [6, 7, 13] to replicas of an object supporting abstract methods. Before performing a method op_i on an object o , a

Received March 8, 1999; revised November 11, 1999; accepted January 12, 2000.
Communicated by Pen-Chung Yew.

quorum number N_t of replicas of o are locked. Suppose a pair of methods op_t and op_u are issued to replicas. If op_t and op_u are update methods, then “ $N_t + N_u > a$ ” must be held in the traditional quorum-based protocol. If op_t is performed on one replica, say o^t , and op_u is performed on another replica, o^u , then the states of the replicas o^t and o^u will be different. The replicas o^t and o^u have the same state if op_t is performed on o^u , op_u is performed on o^t , and 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 performed. However, there can exist replicas from which the newest version can be constructed even if there is no newest replica. In order to do this, we have to identify which methods are performed on each replica. We newly propose a *version vector* to identify the methods performed on each replica. In the QOL protocol, fewer replicas are locked than in the quorum-based protocol or the 2PL protocol.

In section 2, we extend the traditional quorum concepts to the object-based system. In section 3, we present the QOL protocol. In section 4, we evaluate the QOL protocol.

2. OBJECT QUORUMS

2.1 Objects

A system is composed of objects interconnected in a reliable network. Each object o is encapsulated so that o can be manipulated only by using a method op_t ($t = 1, \dots, l$) supported by the object o . For example, a *counter* object supports the methods *up* and *down*, which increment and decrement the counter, respectively. The *counter* object is allowed to be updated only by the methods *up* and *down*. In the network, messages can be delivered to their destinations with no message loss in the sending order.

The objects are distributed on multiple computers. A *server* computer stores objects. Applications initiate transactions in a *client* computer. Transactions issues requests to the servers to manipulate objects in the servers. If a transaction sends a request for a method op to an object o , then op is performed on o . Then, the response of op is sent back to the transaction. A transaction is an atomic invocation sequence of methods. The transaction *commits* only if all the methods invoked by the transaction successfully complete, i.e. commit. The method op is also an atomic unit of computation in the server. A method which changes the state of the object is referred to as an *update* method.

Let $op(s)$ denote a state obtained by performing a method op on a state s of an object o . A method op_t is *compatible* with another method op_u iff $op_t \circ op_u(s) = op_u \circ op_t(s)$ for every state s of o . Otherwise, op_t *conflicts* with op_u . The conflicting relation C among the methods is not transitive. We assume that the conflicting relation C is symmetric. For example, *read* conflicts with *write*, and *write* conflicts with *read*. The conflicting relation is specified on defining the object o based on the meaning of the object o . The interleaved and parallel computation of methods has to be *serializable* [2]. That is, if some method op_1 of a transaction T precedes a method of another transaction T' which conflicts with op_1 , then every method op_2 of T precedes a method of T' which conflicts with op_2 . All transactions are required to be totally ordered in precedent order to realize serializability. A method op is *absorbing* iff $op_1 \circ op(s) = op(s)$ for every method op_1 and every state s of the object o . For example, *write* is an absorbing method.

Example 1. A *Counter* object I supports four types of methods, *Up*, *Down*, *Check*, and *Log*, which are abbreviated as Up , Dwn , Chk , and Lg , respectively. The methods Up and Dwn increment and decrement the value of the counter I , respectively. The method Chk reads the counter. The counter has a log where the methods Up and Dwn performed on I are recorded. The method Lg manipulates the log in the counter. The method Up is compatible with the method Dwn . Chk is compatible with Lg because Chk reads the counter I and Lg manipulates the log but does not manipulate the counter. Up and Dwn conflict with Chk and Lg . Lg conflicts with itself. The methods Up , Dwn , and Lg are update methods, but Chk is not. Fig. 1 presents a graph showing the conflicting relations among the methods. Here, a node shows a method, and an edge between the nodes indicates that two methods denoted by the nodes conflict. \square

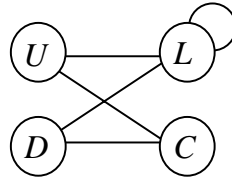


Fig. 1. Conflicting graph.

On receipt of a request op_t , an object o is locked in a *lock mode* $\mu(op_t)$ in order to make the computation serializable. If op_t is compatible with op_u , then the mode $\mu(op_t)$ is compatible with the mode $\mu(op_u)$. Otherwise, $\mu(op_t)$ conflicts with $\mu(op_u)$. For example, *Ulock* and *Dlock* denote lock modes of the methods Up and Dwn of the *Counter* object I , respectively. *Ulock* and *Dlock* are compatible. After performing op_t , the lock of the mode $\mu(op_t)$ on the object o is released.

2.2 Quorum Constraints

Let R be a *cluster*, i.e. a set of replicas o^1, \dots, o^a of an object $o(a \geq 1)$. We extend the traditional quorum-based protocol [7] to lock replicas of objects in the object-based system. Let Q_t be a subset of the replicas to be locked by a method op_t , named a *quorum* of op_t ($Q_t \subseteq R$) (for $t = 1, \dots, l$). Let $N_t (= |Q_t|)$ be the *quorum number* of op_t . The quorums have to satisfy the following object-based quorum (*OBQ*) constraint.

OBQ constraints

- $Q_1 \cup Q_l = R$.
- If $\mu(op_t)$ conflicts with $\mu(op_u)$, then $N_t + N_u > a$, i.e. $Q_t \cap Q_u \neq \emptyset$. \square

The quorum numbers N_1, \dots, N_l are decided so that the more frequently each method op_t is invoked, the fewer replicas are locked by op_t .

A transaction T locks the replicas of the object o by means of the following locking protocol before manipulating the replicas using a method op_t .

- 1 First, a quorum Q_t for op_t is constructed by selecting N_t replicas in the cluster R , $Q_t \subseteq R$.
- 2 Every replica in Q_t is locked in a mode $\mu(op_t)$.
- 3 If all the replicas in Q_t are locked, then the replicas in Q_t are manipulated by op_t .
- 4 When T commits, the locks on the replicas in Q_t are released.

According to the traditional quorum-based protocols, $N_t + N_u > a$ if one of the methods op_t and op_u is an update type method. On the other hand, the OBQ constraints mean that $N_t + N_u > a$ only if op_t conflicts with op_u . In other words, " $Q_t \cap Q_u = \phi$ " can hold even if op_t or op_u is an update method. The OBQ constraints satisfy the following properties.

Object-Based Quorum (OBQ) properties: For every pair of conflicting methods op_t and op_u of an object o :

- 1 Both methods op_t and op_u performed on at least $k(= N_t + N_u - a)$ replicas.
- 2 op_t and op_u are performed on every pair of common destination replicas in the same order. □

2.3 Precedency Among Replicas

Example 2. Suppose there are four replicas I^1, I^2, I^3 , and I^4 of the *Counter* object I presented in Example 1, where $a = 4$. Each replica I^t has a version number V^t whose initial value is 0 ($t = 1, \dots, 4$) according to the traditional protocol. Suppose that the quorum number N_{Up} for Up is 3, and that N_{Dwn} is 2. Here, $N_{Up} + N_{Dwn} > 4$. Up is issued to three replicas, say I^1, I^2 , and I^3 , and the version numbers are updated as $V^1 = V^2 = V^3 = 1$. Then, Dwn is issued to I^1 and I^4 . Since $V^1(= 1) > V^4(= 0)$, Dwn is performed on I^1 and V^1 is changed to 2. I^4 is updated by taking the state from I^1 . Here, $V^1 = V^4 = 2$ and $V^2 = V^3 = 1$. If the quorum number is decided based on the conflicting relations among the methods according to the object-based quorum (OBQ) constraints, we can reduce the quorum number but cannot decide which replica is newest by using the version numbers. Here, $N_t + N_u > a$ if a pair of methods op_t and op_u conflict. For example, N_{Up}, N_{Dwn}, N_{Chk} , and N_{Lg} can be 2, 2, 3, and 3, respectively [Fig. 2]. First, suppose Up is issued to two replicas I^1 and I^2 , and that Dwn is issued to I^3 and I^4 since $N_{Up} = N_{Dwn} = 2$ [step 1 of Fig. 2]. Here, the version numbers of the replicas are changed to 1, i.e. $V^1 = V^2 = V^3 = V^4 = 1$. Then, the method Lg is issued to the replicas I^1, I^2 , and I^3 since $N_{Lg} = 3$. Lg conflicts with Up and Dwn as shown in Fig. 1. The state of I^3 is different from those of I^1 and I^2 although I^1 and I^2 have the same version number $V^1 = V^2 = 1$. The problem is that the states of I^1 and I^2 cannot be recognized as being different from those of I^3 and I^4 by using the version numbers. Since Up and Dwn are compatible, the instance of the method Up performed on the replicas I^1 and I^2 is also performed on I^3 and I^4 , and the instance of Dwn performed on I^3 and I^4 is performed on I^1 and I^2 [step 2 in Fig. 2]. Then, Lg can be performed on the replicas I^1, I^2 , and I^3 [step 3 in Fig. 2]. Thus, the replicas can be made to be the newest ones by performing methods which are performed not on the replicas but on the others if these methods are compatible. □

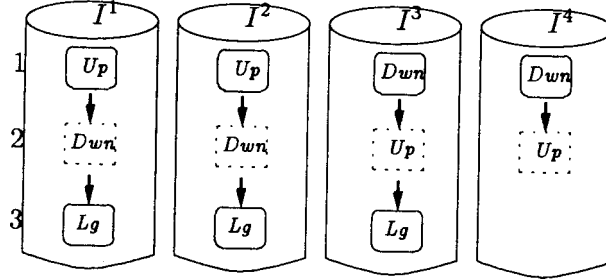


Fig. 2. Example 2.

In Example 2, the replicas I^1 , I^2 , and I^3 are *newer* than I^4 after Up , Dwn , and Lg are performed because Lg is not performed on I^4 . However, it is not decided which one I^1 and I^3 is newer after Up and Dwn are performed. A replica o^h is considered to be *newer* than another replica o^k if every method instance performed on o^k is performed on o^h . In the traditional protocol, a replica o^h is *newer* than o^k if every *write* performed on o^k is performed on o^h . We define a precedent relation “ \rightarrow ” among replicas to show which replica is newer.

Definition 1: A replica o^h *precedes* another replica o^k ($o^h \rightarrow o^k$) iff for each method op performed on o^k , op is performed on o^h if some method conflicting with op is performed on o^h and every method conflicting with op and performed on o^k is performed on o^h . \square

The precedent relation “ $o^h \rightarrow o^k$ ” means that o^h is newer than a replica o^k since some methods performed on o^h which conflict with other methods are not performed on o^k . In Example 2, I^1 , I^2 , and I^3 precede I^4 ($I^1 \rightarrow I^4$) in step 3 in Fig. 2. A replica o^h is *maximal* iff there is no replica o^k such that $o^h \rightarrow o^k$ in the cluster R . o^h is *maximum* iff $o^h \rightarrow o^k$ for every replica o^k in R . In Example 2, the replicas I^1 , I^2 , I^3 , and I^4 are maximal, but there is no maximum replica in step 1. In step 3, the replicas I^1 , I^2 , and I^3 are maximum. The maximum replica shows the newest replica in the cluster. If there is only one replica in the cluster, the maximum replica shows the same state of the replica. The quorum-based protocol requires that every quorum include at least one maximum replica o^h . That is, if a pair of conflicting methods op_t and op_u are issued, then both op_t and op_u are performed on at least one replica o^h . On the other hand, according to the OBQ constraints, both op_t and op_u may not be performed on any replica if op_t and op_u are compatible even if op_t and op_u are update methods. However, each of op_t and op_u is surely performed on some replica. Hence, there may be no maximum replica, but multiple maximal may exist replicas. A cluster R is *complete* iff there is a maximum replica in R . The cluster R may be incomplete in the QOL protocol although R is always complete in the quorum-based protocol.

As explained in Example 2, the replicas I^1 and I^3 whose states are not the same in step 1 can be *unified* into the same state by exchanging the methods Up performed on I^1 and Dwn performed on I^3 with one another. Let σ_t be a sequence $\langle op_{t1}, \dots, op_{t_l} \rangle$ ($l_t \geq 0$) of methods performed on a replica o^t ($t = 1, \dots, a$). For every pair of replicas o^h and o^k , σ_{hk} and σ_{kh} are *incomplete* postfixes $\langle op_{h1}, \dots, op_{h_{l_h}} \rangle$ ($l_h \geq 1$) and $\langle op_{k1}, \dots, op_{k_{l_k}} \rangle$ ($l_k \geq 1$) of σ_h and σ_k , respectively, where $op_{h,i-1} = op_{k,j-1}$, $op_{h,j-1}$ is an absorbing method and there are no incom-

plete postfixes σ'_{hk} and σ'_{kh} of σ_{hk} and σ_{kh} , respectively. If σ_{hk} and σ_{kh} are empty, then $\sigma_{hk} = \sigma_h$ and $\sigma_{kh} = \sigma_k$. The states of I^h and I^k are the same if $op_{h,i-1}$ and $op_{k,j-1}$ are performed on I^h and I^k , respectively, independently of which methods are performed before $op_{h,i-1}$ and $op_{k,j-1}$. An *incomplete* method is an update method in the incomplete postfix. An update method in the sequence σ , but not in the incomplete postfix is *complete*.

Definition 2: A pair of replicas o^h and o^k are *unifiable* ($o^h \equiv o^k$) iff every method in one of incomplete postfixes σ_{hk} and σ_{kh} is compatible with every method in the other. \square
The unifiable relation “ \equiv ” is equivalent.

If a pair of replicas o^h and o^k are unifiable, then o^h and o^k can be made the same by exchanging the methods in the incomplete postfixes σ_{hk} and σ_{kh} . In Example 2, Up is performed on the replicas I^1 and I^2 , and Dwn is performed on I^3 and I^4 . Here, $\sigma_{13} = \langle Up \rangle$ and $\sigma_{31} = \langle Dwn \rangle$. Up and Dwn is incomplete methods. If Dwn is performed on I^1 and Up is performed on I^3 , I^1 and I^3 are unified. Here, a *least upper bound* of I^1 and I^3 ($I^1 \cup I^3$) shows the state of I^1 and I^3 obtained by exchanging the methods. It is straightforward that $I^1 \cup I^3 \rightarrow I^1$, $I^1 \cup I^3 \rightarrow I^3$, and that there is no state I such that $I^1 \cup I^3 \rightarrow I \rightarrow I^1, I^3$.

Let $UN(o^h)$ be an equivalent set $\{o^k \mid o^k \equiv o^h \text{ in } R\}$ for a maximal replica o^h in a cluster R . In Example 2, the replicas I^1, I^2, I^3 , and I^4 are maximal and $UN(I^1) = \{I^1, I^2, I^3, I^4\} = UN(I^2) = UN(I^3) = UN(I^4)$. A cluster R is *consistent* iff $UN(o^h) = UN(o^k)$ for every pair of maximal replicas o^h and o^k in R . Here, $UN(o^h)$ is referred to as the *unifiable set* UN of the cluster R . A least upper bound of the replicas in a consistent cluster R shows a possible maximum replica to be obtained from the replicas in R . That is, $\cup\{o^k \mid o^k \in R\}$ is a replica which can be made to be the maximum, i.e. the newest one, by exchanging the incomplete methods performed. If the cluster R is inconsistent, then the replicas cannot be consistent.

In an incomplete cluster R , some methods performed on a maximal replica have to be performed later on other maximal replicas on which the methods have not been performed yet. *Incomplete* methods are update ones which are performed on some replicas but not on every replica in a unifiable set UN of the cluster R . In Example 2, as a result of exchanging the incomplete methods Up and Dwn with I^2 and I^1 , I^1 and I^2 become the same. Hence, the replicas I^1 and I^2 are unifiable, i.e. $I^1 \equiv I^2$. $UN = \{I^1, I^2, I^3, I^4\}$. Every pair of incomplete methods not performed on the same replica is performed not on every replica. Complete methods are performed on every replica or absorbed by other complete methods performed on every replica.

Suppose a method op_t is issued to the replicas in the cluster R . The method op_t can be performed on a replica o^h in the quorum Q_t of op_t if every incomplete method conflicting with the method op_t is performed on o^h . However, there might not exist such a replica o^h in the cluster R . Hence, op_t is performed as follows:

1. Incomplete methods on each maximal replica are performed on the other maximal replicas in the quorum Q_t where the methods have not been performed as discussed above. Here, every maximal replica is the maximum, i.e. the newest one.
2. The method op_t is performed on the maximal replicas.
3. If op_t is an update method, then the states of the replicas in Q_t have to be changed. Every update method performed on the maximal replicas but not performed on the other replicas is performed on the non-maximal replicas in the cluster R . In other words, one of the maximal replicas sends the state to the other replicas.

Here, every replica in the quorum Q_t is the newest one, i.e. the maximum replica in R .

Example 3: Initially, each replica I^h of the counter object I has the version vector $VV^h = \langle 0_{0000}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$ for $h = 1, \dots, 4$ in Example 1. Suppose the method Up is issued to two replicas, I^1 and I^2 , since $N_{Up} = 2$. $VV^1 = VV^2 = \langle 1_{1100}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$. Then, the method Dwn is issued to the two replicas I^3 and I^4 since $N_{Dwn} = 2$. $VV^3 = VV^4 = \langle 0_{0000}, 1_{0011}, 0_{0000}, 0_{0000} \rangle$. Then, the method Chk is issued to the three replicas I^1 , I^2 , and I^3 since $N_{Chk} = 3$. $VV^1 = VV^2 \neq VV^3$. Since $V_{Up}^1 = V_{Up}^2 = 1_{1100}$ and $V_{Dwn}^3 = 1_{0011}$, I^1 and I^2 know that Up has been issued to I^1 and I^2 , and I^3 knows that Dwn has been issued to I^3 and I^4 . Here, no replica is maximum because either Up or Dwn has been performed on every replica. One replica, say I^1 , is selected. The instance of Dwn performed on I^3 is performed on I^1 . V_{Dwn}^1 is changed to be 1_{0011} , i.e. $VV^1 = \langle 1_{1100}, 1_{0011}, 0_{0000}, 0_{0000} \rangle$. Then, the method Chk is performed on I^1 . Since Chk is not an update method, V_{Chk}^1 is not changed. Up is issued to I^3 and I^4 . $VV^3 = VV^4 = \langle 1_{0011}, 1_{0011}, 0_{0000}, 0_{0000} \rangle$. Then, Lg is issued to the three replicas I^2 , I^3 , and I^4 since $N_{Lg} = 3$. Since $V_{Up}^2 = 1_{1100}$ and $V_{Up}^3 = 1_{0011}$, different instances Up_1 and Up_2 of the method Up are performed on I^2 and I^3 , respectively. I^2 and I^3 exchange the instances of Up performed with one another. In addition, $V_{Dwn}^2 = 0_{0000}$ and $V_{Dwn}^3 = 1_{0011}$. Here, the instances of Up and Dwn performed on I^3 have to be performed on I^2 to obtain the least upper bound version of I^2 and I^3 . Since Up is compatible with Dwn , Up and Dwn can be performed on I^2 in any order. Now, the method Lg is performed on I^2 after Up and Dwn are performed. Here, a sequence $\langle Up_1, Dwn, Up_2, Lg \rangle$ is performed on I^2 , and $\langle Dwn, Up_2, Up_1, Lg \rangle$ is performed on I^3 . $VV^2 = \langle 2_{1111}, 1_{0011}, 0_{0000}, 1_{0111} \rangle$ and the method Lg updates I^2 . If the state of the replica I^2 is sent to I^3 and I^4 , I^3 and I^4 are updated with the state of I^2 . $VV^2 = VV^3 = VV^4$. Here, $V_{Up}^1 = V_{Up}^2 = V_{Up}^3 = V_{Up}^4 (= 2_{1111})$ is initialized to be 0_{0000} since the same instances of Up are surely performed on every replica. Instead of ending the states, the method Lg can be performed on the replicas I^3 and I^4 . Then, I^2 can send a sequence of the instances performed on I^2 to the other replicas. \square

Let B^h and B^k be bitmaps for the replicas o^h and o^k , respectively. B^h is *included* in B^k ($B^h \subseteq B^k$) iff $b^{hj} = 1$ if $b^{kj} = 1$ for $j = 1, \dots, a$. $B^h \cup B^k$ shows $\langle b^1, \dots, b^a \rangle$, where $b^j = 0$ if $b^{hj} = b^{kj} = 0$ or 1 for $j = 1, \dots, a$.

- $V_t^h \leq V_t^k$ iff $C_t^h \leq C_t^k$ and $B_t^h \subseteq B_t^k$.
- $VV^h \leq VV^k$ iff $V_t^h \leq V_t^k$ for every method op_t .
- $VV^h \leq_t VV^k$ iff $V_t^h \leq V_t^k$ and $V_u^h \leq V_u^k$ for every method op_u conflicting with a method op_t .
- $VV^h \leq_* VV^k$ iff $VV^h \leq_t VV^k$ and $VV^h \leq_u VV^k$ for every pair of conflicting methods op_t and op_u .

A pair of version elements V_t^h and V_t^k are *not uncomparable* ($V_t^h \parallel V_t^k$) iff neither $V_t^h \leq V_t^k$ nor $V_t^h \geq V_t^k$. If $B_t^h \cap B_t^k \neq \emptyset$, then $V_t^h \parallel V_t^k$ even if $C_t^h \leq C_t^k$ or $C_t^k \leq C_t^h$. For example, suppose that $VV^1 = \langle 1_{1100}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$ and $VV^3 = \langle 2_{1110}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$. Here, $VV^1 \leq VV^3$. If $VV^2 = \langle 2_{0011}, 0_{0000}, 0_{0000}, 0_{0000} \rangle$, then $V_{Dwn}^1 \parallel V_{Dwn}^2$. In a subset $Q \subseteq R$, VV^h is *maximal* iff there is no replica o^k where $VV^h \leq_* VV^k$. The version vector VV^h of a replica o^h is *maximal* with respect to a method op_t iff there is no replica o^k , where $VV^h \leq_t VV^k$ in the subset Q of the cluster R .

Theorem 1: If $VV^h \leq_* VV^k$, then every pair of instances of conflicting methods performed on a replica o^h is performed on another replica o^k in the same order. \square

Proof: From the definition of \leq_* , it is straightforward that the theorem holds. \square

Theorem 2: A replica o^h is maximal and maximum if VV^h is maximal and maximum, respectively, in a cluster R . \square

Proof: From Theorem 1, $VV^k \leq_* VV^h$ means that every method conflicting with methods performed on a replica o^k is performed on o^h . Hence, the theorem holds. \square

We define a *least upper bound* operation “ \cup ” for a pair of version elements V_t^h and V_t^k on op_t as follows:

$$\bullet V_t^h \cup V_t^k = \begin{cases} V_t^k & \text{if } V_t^h \leq V_t^k. \\ V_t^h & \text{if } V_t^h \geq V_t^k. \\ \langle C_t^h + C_t^k, B_t^h \cup B_t^k \rangle, & \\ \text{otherwise.} & \end{cases}$$

$$\bullet VV^h \cup VV^k = \langle V_1^h \cup V_1^k, \dots, V_t^h \cup V_t^k \rangle.$$

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

Definition 3: A version vector VV^h is *equivalent* with another version vector VV^k ($VV^h \equiv VV^k$) iff $V_v^h = V_v^k$ for every update method op_v conflicting with every pair of compatible methods op_t and op_u . \square

For example, the methods *Up* and *Dwn* are compatible with one another and conflict with the method *Lg* in Example 3. Hence, $VV^1 = \langle 1_{1100}, 0_{0000}, 0_{0000}, 2_{0101} \rangle$ is equivalent to $VV^2 = \langle 0_{0000}, 1_{0011}, 0_{0000}, 2_{0101} \rangle$ ($VV^1 \equiv VV^2$) since $V_{Lg}^1 = V_{Lg}^2 = 2_{0101}$. The replicas I^1 and I^2 are the same if the instances of *Dwn* and *Up* are performed on I^1 and I^2 , respectively. Thus, if $VV^h \equiv VV^k$, then the replicas o^h and o^k can be made the same by performing compatible methods which have not yet been performed on the replicas.

3.2 Protocol

We will next discuss a locking protocol using the version vector. Suppose a method op_t is issued to an object o . A replica o^h of the object o has a *method log* l^h for storing a sequence σ_h of method instances performed on o^h . Let l_t^h be a sequence of instances of the method op_t in the method log l^h . Each instance op has a bitmap $op.B = \langle b^1, \dots, b^a \rangle$ showing replicas to which op is issued. That is, $op.b^k = 1$ if op is issued to a replica o^k . The counter C_t^h gives the number of instances in l_t^h . For example, $V^2 = \langle 2_{1111}, 1_{0011}, 0_{0000}, 1_{0111} \rangle$ in Example 3 shows that two instances Up_1 and Up_2 of the method *Up* are performed on every replica, one instance of *Dwn* is performed on two replicas I^3 and I^4 , and one instance of *Lg* is performed on three replicas I^2 , I^3 , and I^4 .

QOL protocol: A client c sends a method op_t to every replica in the quorum Q_t of op_t .

1. All the replicas in the quorum Q_t are locked in a mode $\mu(op_t)$. Each replica o^h in Q_t sends to the client c a response with the version vector VV^h and the method \log^h . Unless it succeeds in locking the replicas, the method op_t aborts.
2. On receipt of the responses from all the replicas in the quorum Q_t , the client c obtains a least upper bound of the version vectors $VV_s = \cup\{VV^k \mid o^k \in Q_t\}$. Let $P_h(op_t)$ denote a set $\{op_u \mid op_u \text{ conflicts with } op_t, op_u.B \neq \langle 1\dots 1 \rangle, \text{ and } op_u \text{ is performed on } o^h\}$. The client c finds a replica o^h in Q_t which is maximal with respect to methods conflicting with op_t , i. e. $VV^l \preceq VV^h$ for every replica o^l in Q_t .
3. If a replica o^h is found, then the method op_t is performed on o^h .
 - a. If op_t is not an update method, then o^h sends a response to the client c .
 - b. Otherwise, o^h sends the set $P_h(op_t)$ to one replica o^k in Q_t . Every method op_u in $P_h(op_t)$ is performed on o^k unless op_u has been performed on o^k . For every op_v in $P_h(op_t)$, $op_v.B := op_v.B \vee op_t.B$. o^k sends a response to the client c .
4. Unless o^h is found, the client c selects one maximal replica o^h in the quorum Q_t . Let $P(op_t)$ be a set $\{op_u \mid op_u \text{ conflicts with } op_t \text{ and is performed on some replica } o^h \text{ in } Q_t\}$.
 - a. If op_t is an update method, then the client c sends $P(op_t)$ to every replica in Q_t . Each replica o^h performs every update method op_u performed in Q_t which is not performed on o^h and then performs op_t . For every method op_u in $P(op_t)$, $op_u.B := op_u.B \vee op_t.B$ in o^h . o^h sends a response to the client c .
 - b. If op_t is not an update method, then the client c selects one maximal replica o^h in Q_t . The client c sends $P(op_t)$ to o^h . Every op_u in $P(op_t)$ is performed on o^h if op_u has not been performed on o^h . Then, o^h performs op_t . o^h sends a response to the client c . \square

Methods stored in the method \log^h of a replica o^h eventually have to be removed in order to reduce the log size. The bitmap $op_t.B$ attached to a method op_t in the log l^h means that the replica o^h knows that the instance op_t is performed on the replica o^k if $b^k = 1$. If $op_t.B = \langle 1\dots 1 \rangle$, then o^h knows that op_t is performed on every replica. However, op_t cannot be removed from the log l^h in o^h because another replica o^k may not yet know that op_t has been performed on every replica. Hence, a following instance op_u in the log l^h is removed:

- $op_u.B = op_v.B = \langle 1\dots 1 \rangle$ for every method op_v in the log l^h which conflicts with op_u and is performed before op_u .

The counter C_t^h and the bitmap B_t^h are initialized again, i.e. $C_t^h := 0$ and $B_t^h := \langle 0\dots 0 \rangle$ if B_t^h gets $\langle 1\dots 1 \rangle$. If $BM_t^h = \langle 1\dots 1 \rangle$ in some replica o^h , then the method op_t is performed on every replica o^k in the quorum Q_t . Thus, C_t^h shows how many instances of op_t are performed on the replica o^h . If $B_t^h \cap B_t^k = \emptyset$ and ($C_t^h > 0$ or $C_t^k > 0$) for an update method op_t , then a sequence σ^h of instances of op_t performed on o^h is different from a sequence σ^k of o^k . The sequences σ^h and σ^k include the C_t^h and C_t^k instances of op_t , respectively. In the QOL protocol, σ^k and σ^h are required to be performed on o^h and o^k , respectively. Then, $B_t^h := B_t^k := B_t^h \cap B_t^k$ and $C_t^h := C_t^k := C_t^h + C_t^k + 1$.

The following properties hold for the version vectors.

- For every update method op_t , $B_t^h = B_t^k$ and $C_t^h = C_t^k$ if $B_t^h \cap B_t^k \neq \emptyset$ in some pair of replicas o^h and o^k .
- For every update method op_t , $C_t^h \leq C_t^k$ if $B_t^h \subseteq B_t^k$ in some pair of replicas o^h and o^k .
- For every update method op_t , if $B_t^h \subseteq B_t^k$ and $C_t^h \leq C_t^k$, then every instance of op_t performed on the replica o^h is also performed on o^k .

Theorem 3: Every pair of maximal replicas o^h and o^k is unifiable to one unique replica in the QOL protocol. \square

Proof: Assume that a pair of methods op_t and op_u conflict. If op_t is performed on the replica o^h , then op_u is also performed on o^h . Otherwise, o^h is not maximal. Here, both op_t and op_u are performed on o^h or o^k or on neither of them from the QOL properties. Instances performed on either o^h or o^k do not conflict. These methods can be performed in any order. Therefore, o^h and o^k get the same one if the instances performed on one replica are performed on the other replica. \square

The following theorem holds from the above theorems.

Theorem 4: All the replicas are unifiable with one unique replica in the QOL protocol. \square

Proof: From Theorems 2 and 3, it is straightforward that theorem holds.

4. EVALUATION

4.1 Reliability

Let R be a cluster $\{o^1, \dots, o^a\}$ of an object o ($a \geq 1$). In the quorum-based protocol, N_w and N_r replicas in a cluster R of an object o are locked for read and write methods. Let k be $N_w + N_r - a$. As long as fewer than k replicas are faulty, at least one replica o' is surely newest; i.e. all read and write methods are performed on the replica o' . Hence the traditional *quorum-based* protocol supports k -resiliency. There is no intersection of the quorum sets of compatible methods in the QOL protocol. For example, the methods Up and Dwn (*Down*) of the counter object I are compatible as shown in Example 1, where the quorum numbers N_{Up} and N_{Dwn} are 2. On the other hand, $N_{Up} = 2$ and $N_{Dwn} = 3$ in the *quorum-based* protocol. Hence, there is at worst only one newest replica. If the newest replica is faulty, the system is faulty because the current state of the object is lost. In the QOL protocol, there is no problem even if one replica is faulty. Suppose $Q_{Up} = \{I^1, I^2\}$ and $Q_{Dwn} = \{I^2, I^3\}$ in Example 2. Suppose I^1 is faulty. The operational replicas, say I^2, I^3 , and I^4 , can obtain all the instances performed on the replicas because I^2 is still operational and every instance of the method Up performed on I^1 and I^2 is logged in I^2 .

4.2 Number of Locks

We evaluated the QOL protocol by comparing it with the traditional quorum-based protocol in terms of the number of the replicas locked. An object o supports the methods $op_1, \dots, op_l (l \geq 1)$. In the quorum-based protocol, an update method op_i of an object o is considered to be *write*. Otherwise, op_i is *read*. Quorums Q_i and Q_u for a pair of methods op_i and op_u , respectively, are decided so that $Q_i \cap Q_u \neq \emptyset$ if op_i or op_u is an update method. On the other hand, the quorums for the methods are obtained based on the conflicting relations among the methods. $Q_i \cap Q_u \neq \emptyset$ only if op_i conflicts with op_u . That is, $Q_i \cap Q_u \neq \emptyset$ if op_i is compatible with op_u even if op_i or op_u is an update method in the QOL protocol. Let $\varphi(op_i)$ denote a usage frequency of a method op_i where $\varphi(op_1) + \dots + \varphi(op_l) = 1$. Let C denote a conflicting relation where $C_{iu} = 1$ if op_i conflicts with op_u ; otherwise, $C_{iu} = 0$. For various values of the usage frequency φ and the conflicting relation C , the minimum quorum numbers N_i^o and N_u^o to be locked in the QOL protocol and the quorum-based protocol, respectively, are calculated.

In the evaluation, an object was assumed to support two types of methods op_1 and op_2 . Fig. 4 shows conflicting graphs for three cases, i.e. case 1: op_1 conflicts with op_1 and op_2 , and case 2: op_1 is compatible with op_2 .

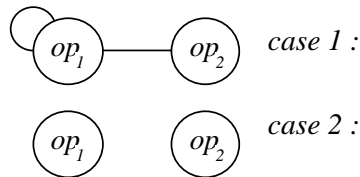


Fig. 4. Conflicting relations.

Figs. 5 and 6 show the quorum number for the usage frequency of op_1 for the case in which the number a of replicas is 10. In the quorum-based protocol, the quorum number of the method op depends on whether or not op is an update method. We assume that op is an update method if op conflicts with itself. We also assume that one of the methods, op_1 or op_2 , is an update method if op_1 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. There are three cases depending on whether or not the methods are update ones. Hence, there are two combinations of update and non-update methods in case 1 and four in case 2. We obtained the average quorum numbers for the quorum-based protocol through computation. The quorum number of the QOL protocol was also calculated so that the object-based quorum (OBQ) constraints would be satisfied. Figs. 5 and 6 show that fewer replicas are locked in the QOL protocol than in the quorum-based protocol. Even if a pair of methods, op_1 and op_2 are update methods, op_1 may be compatible with op_2 . For example, the method Up is compatible with the method Dwn in Example 1.

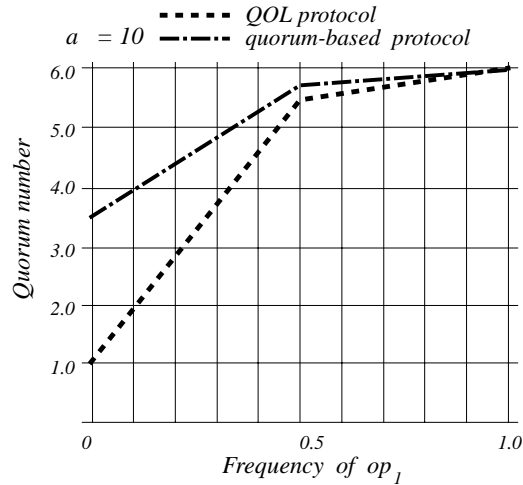


Fig. 5. Evaluation of the QOL protocol (case 1).

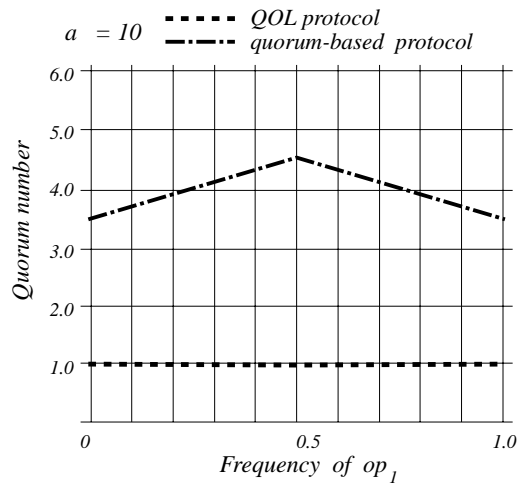


Fig. 6. Evaluation of the QOL protocol (case 2).

5. CONCLUSIONS

This paper has discussed the *quorum-based object locking (QOL)* protocol for the replicas of objects. It extends the quorum-based protocol with *read* and *write* methods, to the object-based system with abstract methods. The object supports a more abstract level of method than do read and write. The version vector has been proposed to maintain mutual consistency among the replicas. The replicas are not required to perform every update method instance which has been computed on the other replicas if the instance is compatible with the instances performed. Through the evaluation, we have shown that by using the QOL protocol, the more efficient access to replicated objects can be realized in the dis-

tributed system, i.e. fewer number of replicas are locked than the traditional quorum-based protocol. Furthermore, the QOL supports higher level of reliability and availability against replica faults than the quorum-based protocol.

ACKNOWLEDGMENT

The authors would like to thank Dr. Hiroaki Higaki, Tokyo Denki Univ., for helpful discussions and comments on this paper.

REFERENCES

1. M. Ahamad, P. Dasgupta, R. LeBlanc, and Wilkes, "Fault tolerant computing in object based distributed operating systems," in *Proceedings of the 6th IEEE Symposium on Reliable Distributed Systems*, 1987, pp. 115-125.
2. P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison Wesley, 1987.
3. P. A. Bernstein and N. Goodman, "The failure and recovery problem for replicated databases," in *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, 1983, pp. 114-122.
4. J. M. Carey and M. Livny, "Conflict detection tradeoffs for replicated data," *ACM Transactions on Database Systems*, Vol. 16, No. 4, 1991, pp. 703-746.
5. P. Y. Chevalier, "A replicated object server for a distributed object-oriented system," in *Proceedings of the 11th IEEE Symposium on Reliable Distributed Systems*, 1992, pp. 4-11.
6. X. Defago, A. Schiper, and N. Sergent, "Semi-passive replication," in *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998, pp. 43-50.
7. H. Garcia-Molina and D. Barbara, "How to assign votes in a distributed system," *Journal of ACM*, Vol. 32, No. 4, 1985, pp. 841-860.
8. K. Hasegawa and M. Takizawa, "Optimistic concurrency control for replicated objects," in *Proceedings of Int'l Symposium on Communications*, 1997, pp. 149-152.
9. K. Hasegawa, H. Higaki, and M. Takizawa, "Object replication using version vector," in *Proceedings of the 6th IEEE Int'l Conference on Parallel and Distributed Systems*, 1998, pp. 147-154.
10. J. Jing, O. Bukhres, and A. Elmagarmid, "Distributed lock management for mobile transactions," in *Proceedings of the 15th IEEE Int'l Conference on Distributed Computing Systems*, Vol. 15, 1995, pp. 118-125.
11. T. Kanazuka, T., H. Higaki, and M. Takizawa, "Quality-based compensation of multimedia objects," in *Proceedings of the 2nd IEEE Int'l Symposium on Object-oriented Real-time Computing*, 1999, pp. 195-202.
12. H. F. Korth, "Locking primitives in a database system," *Journal of ACM*, Vol. 30, No. 1, 1983, pp. 55-79.
13. D. Malkhi and K. M. Reiter, "Secure and scalable replication in Phalanx," in *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 1998, pp. 51-58.

14. M. Silvano and C. S. Douglas, "Constructing reliable distributed communication systems with CORBA," *IEEE Communications Magazine*, Vol. 35, No. 2, 1997, pp. 56-60.
15. T. Yoshida and M. Takizawa, "Model of mobile objects," in *Proceedings of the 7th Int'l Conference on Database and Expert Systems Applications* (Lecture Notes in Computer Science, No. 1134, Springer-Verlag), 1996, pp. 623-632.



Katsuya Tanaka (田中 勝也) was born in 1971. He received his B. E. and M. E. degrees in Computers and Systems Engineering from Tokyo Denki University, Japan in 1995 and 1997, respectively. From 1997 to 1999, he worked for NTT Data Corporation. Currently, he is an assistant in the Department of Computers and Systems Engineering, Tokyo Denki University. His research interest includes distributed systems, transaction management, recovery protocols, and computer network protocols. He is a member of IEEE and IPSJ.



Kyoji Hasegawa (長谷川 享二) was born in 1974. He received his B.E. and M.E. degrees in Computers and Systems Engineering from Tokyo Denki University, Japan in 1997 and 1999, respectively. Currently, he works for Mitsubishi Electric Corporation. His research interests include distributed database system.



Makoto Takizawa (滝沢 誠) was born in Tokyo, Japan, 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 University in 1984. From 1975 to 1986, he worked for Japan Information Processing Developing Center (JIPDEC) supported by MITI. He is currently a full professor of Department of Computers and Systems Engineering and also a director of Information Division of Research Institute for Technology, Tokyo Denki University. From 1989 to 1990, he was a visiting professor of the GMD-IPSI, Germany. He is also a regular visiting professor of Department of Computer Science of Keele University, England since 1990. He was a program co-chair of IEEE ICDCS-18 held in 1998. He is serving on the program committees of many international conferences like ICPADS, ICOIN, ICNP, DEXA, ISORC. He chaired IPSJ SIGDPS and is now a member of the executive board of IPSJ. His research interest includes communication protocols, group communication, distributed database systems, transaction management, and groupware. He is a member of IEEE, ACM, IPSJ, and IEICE.