Invocation Protocol for Replicas in Distributed Object-based Systems

Katsuya Tanaka and Makoto Takizawa

Tokyo Denki University
Email {katsu, taki}@takilab.k.dendai.ac.jp

Object-based systems are composed of objects, which are encapsulations of data and methods. Objects are replicated in order to increase reliability and availability of the systems. Objects are manipulated only through the methods. In addition, methods are invoked in a nested manner. We discuss how to lock replicas of an object for methods in a quorum-based scheme and perform methods on the replicas in presence of nested invocation. First, we extend the quorum concept for primitive read and write methods on simple objects like files to abstract methods supported by objects. The quorum size is decided based on a conflicting relation among methods and state updatability of methods. We present a quorum-based protocol for locking replicas of objects.

オブジェクトの多重化環境におけるメソッドの呼出し方式

田中 勝也　滝沢 誠

東京電機大学

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

高信頼分散オブジェクトシステムでは、複数のレプリカに多重化されたオブジェクトが、メソッドの呼出しにより協調動作を行なう。これまでに、read や write のような基本演算に基づき、効果的にレプリカ間の一貫性を保証する方式として、コーラムに基づく方式が研究されている。一方、各オブジェクトが提供するメソッドは、複数の read や write から構築されるより抽象的な手続きである。本研究では、従来のコーラム方式をオブジェクトの抽象演算に基づく方式へと拡張を行なう。さらに、本方式により、従来方式よりもロックされるレプリカ数が削減できることを示す。

1 Introduction

Various kinds of applications are realized in an object-based framework like CORBA [15]. Objects are replicated in order to increase the reliability and availability of an object-based system. Two-phase locking (2PL) protocols [7] and quorum-based protocols [10] are so far discussed to lock replicas. In the two-phase locking protocol, one replica is locked for a *read* method and all the replicas are locked for a *write* method. On the other hand, *quorum* numbers N_r and N_w of the replicas are locked for *read* and *write*, respectively, in the quorum-based protocol [10,11] where $N_r + N_w > a$ for the number a of the replicas. The subset of the replicas is a *quorum*.

An object is an encapsulation of data and methods for manipulating the data. The object is allowed to be manipulated only through the methods. Methods are more abstract than primitive methods *read* and *write* on a simple object like file. A pair of methods *conflict* on an object if the result obtained by performing the methods depends on the computation order of the methods. The methods are *compatible* if they do not conflict. For example, *increment* and *decrement* methods are compatible on a *counter* object. In the papers [16, 17], the quorum concept for *read* and *write* is extended to abstract methods. Suppose a pair of methods t and u are issued to replicas x_1 and x_2 of an object x. The method t is performed on one replica x_1 and the other method u on another replica x_2 if t and u do not conflict. Here, states of the replicas x_1 and x_2 are different because u is not performed on x_1 and t is not performed on x_2. The replicas x_1 and x_2 can be the same ones if u is performed on x_1 and t is performed on x_2. As long as only compatible methods t and u are issued, the methods are performed on replicas in their quorums. If some method v conflicting with t is issued to a replica x_1, every instance of t so far performed on another replica is required to be performed on x_1. Even if a replica is updated by a method t or u, $N_t + N_u \leq a$ only if t and u do not conflict. The *exchanging protocol* is discussed to exchange compatible methods among the replicas. However, the protocol is complex and implies larger communication overhead to exchange methods. In this paper, we propose another quorum-based protocol to lock replicas for performing abstract methods. In the protocol, no exchanging protocol is used.

First, these abstract methods supported by objects are classified with respect to two points, whether or not methods derive data from objects and methods change states of objects. In addition, we discuss how states are changed by methods. In one type, an object is updated by using the current state of the object. An *increment* method is an example. In the other type, an object is updated independently of the current state. A *reset* method is an example of this type. Then, we define a quorum number for each method based on these types of methods. Finally, we present a quorum-based protocol for abstract methods to lock replicas of objects. The protocol supports three ways to lock replicas and perform methods on replicas depending on types of methods.

In section 2, we overview the quorum-based protocol for replicas of objects. In sections 3, we classify abstract methods supported by objects.

In section 4, we discuss a protocol.

2 Quorum-based Replication of Object

An object is an encapsulation of data and abstract methods. Let us consider a *counter* object c which supports four types of methods *reset* (*res*), *increment* (*inc*), *decrement* (*dec*), and *display* (*dsp*). A *counter* value is incremented and decremented by methods *inc* and *dec*, respectively. A counter value is displayed by *dsp*. By performing a method *res*, a value of the *counter* object c is initialized to be zero. Suppose there are four replicas c_1, c_2, c_3, and c_4 of the object c. Methods *res*, *inc*, and *dec* are traditionally considered to be *write* methods because the state of the *counter* object c is changed by the methods. *dsp* is a *read* method. Hence, $N_{res} + N_{inc} > 4$, $N_{res} + N_{dec} > 4$, $N_{res} + N_{dsp} > 4$, $N_{inc} + N_{dec} > 4$, $N_{dsp} + N_{inc} > 4$, and $N_{dsp} + N_{dec} > 4$ according to the traditional quorum-based protocols [10]. For example, $N_{res} = N_{inc} = N_{dec} = 3$ and $N_{dsp} = 2$.

The quorum concept for primitive methods *read* and *write* [10] is extended to methods of objects [16, 17]. Here, a pair of methods t and u are referred to as *conflict* on an object iff a result obtained by performing t and u on the object depends on the computation order of the methods [3]. Otherwise, t and u are *compatible*. In the *counter* object, *res* conflicts with all the other methods *inc*, *dec*, and *dsp*. *inc* and *dec* are compatible but *inc* and *dec* conflict with *dsp* and *res*. *dsp* is compatible with itself.

[Object-based quorum (OBQ) constraint] If a pair of methods t and u conflict, $N_t + N_u > a$ where a is the total number of the replicas. □

It is noted that $N_t + N_u \leq a$ only if a pair of methods t and u are compatible even if t or u is an update type. Every pair of conflicting methods t and u of an object x are performed on at least k ($= N_t + N_u - a$) replicas in the same order. $N_{inc} + N_{dec} \leq 4$, e.g. $N_{inc} = N_{dec} = 2$ because *inc* and *dec* are compatible. Suppose $Q_{inc} = \{c_1, c_2\}$ and $Q_{dec} = \{c_3, c_4\}$. Since either *inc* or *dec* is performed on each replica in the quorum, the states of the replicas in Q_{inc} are different from Q_{dec}. However, if *dec* is performed on c_1 and c_2 and *inc* is performed on c_3 and c_4, all the replicas can be the same. This is an *exchanging procedure* where every method t performed on one replica is sent to other replicas where t has not been performed and only methods compatible with t have been performed. Suppose a method *dsp* is issued to three replicas c_1, c_2, and c_3 where $Q_{dsp} = \{c_1, c_2, c_3\}$. Since *dsp* conflicts with *inc* and *dec*, *dsp* cannot be performed on any replica in Q_{dsp} because only *inc* has been performed on replicas c_1 and c_2 and only *dec* has been performed on c_3 as shown at step 1 of Figure 1. Before performing *dsp*, *dec* is performed on c_1 and c_2 and *inc* on c_3. *inc* and *dec* can be performed in any order because they are compatible. Here, c_1, c_2, and c_3 get the same at step 2. *dsp* is performed on c_1, c_2, and c_3 at step 3.

Hence, the exchanging procedure implies large amount of overhead. Methods performed on one replica are required to be transmitted to other replicas where the methods have not been performed.

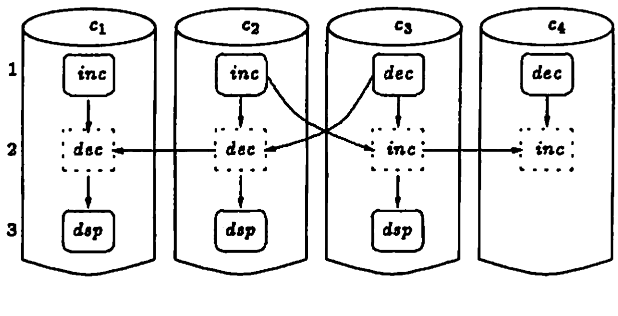


Figure 1: Exchanging procedure.

3 Types of Methods

Objects support abstract levels of methods which are procedures for manipulating the objects. Methods are realized to be procedures which may be implemented by using primitive *read* and *write* methods. The methods are more complex than primitive methods *read* and *write* on simple objects like files and tables [9]. For example, a *counter* object c presented in the preceding subsection supports four types of methods *res* (reset), *inc* (increment), *dec* (decrement), and *dsp* (display). The state of the *counter* object c is changed by the methods *inc*, *dec*, and *res* but *dsp* does not change the object c. Data is derived from the object c by the method *dsp* but no data is derived from c by the other methods *inc*, *dec*, and *res*. A new state obtained by performing *inc* and *dec* on a current state of the *counter* object c depends on the current state. However, a new state obtained by performing *res* is independent of the current state of the object c. That is, any state of the *counter* object c is initialized to be zero by performing *res*. Thus, there are some types of methods. We classify each abstract method t supported by an object o with respect to following points, state type (*stype*), state-dependency type (*dtype*), and output type (*otype*) [Figure 2]:

1. By performing a method t on an object o, a state of the object o is changed if $stype(t)$ is Y. Otherwise, $stype(t)$ is N.
2. If a state $t(s)$ obtained by performing the method t on a current state s of an object o depends on the current state s, $dtype(t)$ is Y. Otherwise, $dtype(t)$ is N.
3. By performing a method t, if some data is derived from the object o and output, $otype(t)$ is Y. Otherwise, $otype(t)$ is N.

Here, it is trivial $stype(t) = Y$ if $dtype(t) = Y$.

For example, four methods *res*, *inc*, *dec*, and *dsp* supported by a *counter* object c are classified as shown in Table 1. For example, any state of the object c is changed with initial value 0 by the method *res*. Hence, $stype(res) = Y$. The state of the *counter* object c obtained by performing *res* on a current state of c is independent of the current state. Hence, $dtype(res) = N$. Since data is not derived by *res*, $otype(res) = N$. A new state

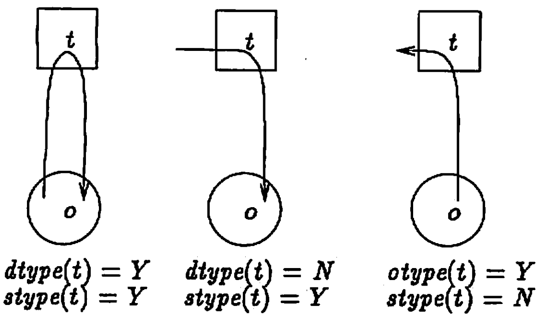


Figure 2: Types of methods.

obtained by performing methods *inc* and *dec* on a current state of the *counter* object c depends on the current state. Hence, $dtype(inc) = dtype(dec) = Y$. $stype(inc) = stype(dec) = Y$. Since no data is output by the methods *inc* and *dec*, $otype(inc) = otype(dec) = N$. On the other hand, a state is not changed by a method *dsp*. Hence, $stype(dsp) = dtype(dsp) = N$ but $otype(dsp) = Y$.

Table 1: Classification of methods.

	stype	*dtype*	*otype*
res	*Y*	*N*	*N*
inc/dec	*Y*	*Y*	*N*
dsp	*N*	*N*	*Y*

Each replica o_i of an object o has a *version counter* ct_i. The version counter ct_i is initially 0. Each time a state of the replica o_i is changed by performing a method, the counter ct_i is incremented by one. That is, $ct_i := ct_i + 1$ if a method t where $stype(t) = Y$ is performed on the replica o_i, i.e. the replica o_i is changed by t. The version counter ct_i of a replica o_i shows how many times the replica o_i is updated, i.e. state is changed.

4 Quorum-based Protocol

4.1 Quorum

Based on types of methods discussed in the preceding section, we extend the traditional quorum concept for primitive *read* and *write* methods to abstract methods supported by objects. If a method t is invoked on an object o, the object o is tried to be locked in a mode $\mu(t)$. A compatibility relation among modes of methods is defined as follows:

[Definition] A mode $\mu(t_1)$ is *compatible* with a mode $\mu(t_2)$ iff a method t_1 is compatible with a method t_2 on an object o. □

In a *counter* object c, a method *inc* (increment) is compatible with a method *dec* (decrement). Hence, a mode $\mu(inc)$ is compatible with a mode $\mu(dec)$. $\mu(inc)$ conflicts with $\mu(dsp)$ and $\mu(res)$ since *inc* conflicts with methods *dsp* (display) and *res* (reset).

Here, suppose a method t is issued to an object o. If an object o is not locked by any transaction or o is locked only in modes which are compatible with a mode $\mu(t)$, the object o is locked in the mode $\mu(t)$. Otherwise, the request of the method t is kept waiting in a wait queue.

An object o is replicated. Let a *cluster* $C(o)$ be a set of replicas $o_1, \ldots, o_a$ of the object o. Suppose a method t is invoked on an object o. A lock request with a mode $\mu(t)$ is issued to a subset Q_t of the replicas in the cluster $C(o)$. Q_t is referred to as a *quorum* of the method t. $Q_t \subseteq C(o)$. N_t is a quorum number of the method t, i.e. the number of replicas in Q_t, $N_t = |Q_t|$ ($\leq n$). The quorum numbers for methods satisfy the following properties:

[Quorum properties] Let t_1 and t_2 be a pair of methods supported by an object o. Here, n is the number of replicas of o.

1. $N_{t_1} + N_{t_2} > n$ if the method t_1 conflicts with the method t_2.
2. $N_{t_1} + N_{t_2} > n$ if $stype(t_1) = Y$ and $stype(t_2) = Y$. □

Let Q_{t_1} and Q_{t_2} be quorums of methods t_1 and t_2 for an object o, respectively. According to the quorum properties, $Q_{t_1} \cap Q_{t_2} \neq \phi$ if t_1 conflicts with t_2 or both of t_1 and t_2 are update methods. If a pair of methods t_1 and t_2 are compatible on an object o, $N_{t_1} + N_{t_2} \leq a$. In a *counter* object c, *inc* and *dec* are compatible. The other methods conflict with *inc* and *dec*. Suppose there are four replicas of the *counter* object c. The methods *inc* and *dec* are compatible but a state of the *counter* object c is changed by *inc* and *dec*, i.e. $stype(inc) = stype(dec) = Y$. Hence, $N_{inc} + N_{dec} > 4$. $N_{dsp} + N_{inc} > 4$. For example, $N_{inc} = 3$, $N_{dec} = 3$, and $N_{dsp} = 2$.

4.2 Protocol

We discuss a protocol for invoking methods on replicas of an object. Suppose a method t is invoked on an object o. An invoker of the method t is referred to as *transaction*. Let $C(o)$ be a cluster $\{o_1, \ldots, o_n\}$ of replicas for an object o. Since the method t may invoke other methods, trnasactions are nested. First, a quorum Q_t is constructed for a given quorum number N_t, i.e. $|Q_t| = N_t$ and $Q_t \subseteq C(o)$. In this paper, replicas to be included in the quorum Q_t of a method t are randomly decided each time t is invoked. Then, a lock request is issued to every replica in the quorum Q_t. Replicas in a quorum Q_t are first locked. Then, it is decided on which replica the method is performed. Lastly, if the locks are obtained on replicas, the method t is performed on the replicas. Thus, the protocol is composed of three phases, *locking*, *decision*, and *execution* phases.

[Locking phase]

1. A lock request of a method t is issued to every replica in a quorum Q_t.
2. If a replica o_i in the quorum Q_t is already locked in a mode which conflicts with a mode $\mu(t)$, a response *No* is sent back to the transaction, i.e. invoker of t.
3. Otherwise, the replica o_i in the quorum Q_t is locked in a mode $\mu(t)$. Here, let $L(o_i)$ be a set of methods whose locks are being held on the replica o_i and whose *stype* is Y. That is, methods in $L(o_i)$ are ones by which the state of the replica o_i is changed. The information $\langle ct_i, L(o_i)\rangle$ is sent back in a *Yes* message to the transaction where ct_i is a counter of o_i. □

The transaction, i.e. invoker of the method t waits for responses from all the replicas in the quorum Q_t.

[Decision phase]

1. If *No* is received from a replica, the transaction sends *Abort* to all the replicas which have sent *Yes*. The locks on the replicas are released.
2. If *Yes* is received from every replica in the quorum Q_t, the transaction sends a *Do* message to all the replicas in the quorum Q_t to perform the method t. □

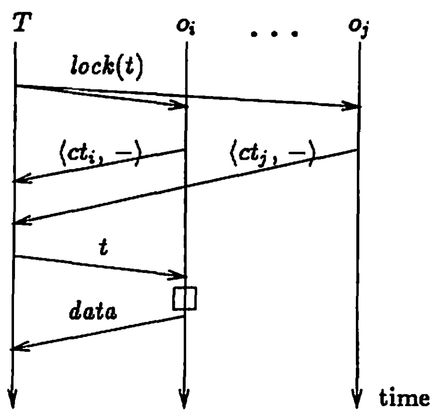
If all the replicas in the quorum Q_t are successfully locked, the method t is tried to be performed on the replicas. It depends on types of methods how to perform the methods on the replicas in the quorum Q_t. There are following types of methods:

1. $otype(t) = Y$ and $stype(t) = dtype(t) = N$.
2. $stype(t) = Y$ and $dtype(t) = N$.
3. $stype(t) = Y$ and $dtype(t) = Y$.

First, let us consider the type 1, i.e. a method t derives the data, but does not change the state of the replica, i.e. $stype(t) = dtype(t) = N$ and $otype(t) = Y$.

[Execution phase for type 1] [Figure 3] Data is derived from an object o but a state of o is not changed by a method t, i.e. $otype(t) = Y$ and $stype(t) = dtype(t) = N$.

1. Let R_t be a subset of the replicas whose version counters are maximum in the quorum Q_t, i.e. $\{o_i \mid o_i \in Q_t \wedge ct_i \geq ct_j$ for every o_j in $Q_t\}$.
2. The transaction sends a *Do* message with a method t to every replica in the subset R_t.
3. The method t is performed on each replica in R_t. Here, the method t might invoke other methods. If the method t eventually completes, a response *Done* with data derived is sent back to the transaction. □



□: computation of t

Figure 3: Type 1.

In the type 2, a state of the object o is changed by a method t, i.e. $stype(t) = Y$. Here, there are two additional cases, $dtype(t) = Y$ or $dtype(t) = N$ depending on whether or not a new state obtained depends on the current state of the object o. If $dtype(t) = N$, a state of the object o is changed with some new state independently of the current state of the object o.

[Execution phase for type 2] [Figure 5] A state of an object o is changed by a method t independently of a current state of o, i.e. $dtype(t) = N$.

1. A method t is issued to every replica in the quorum Q_t.
2. The method t is performed on every replica in the quorum Q_t.
3. If $otype(t) = Y$, the response of t with output data is sent to the transaction. The transaction takes a response from a replica o_i in the quorum Q_t whose version counter ct_i is maximum in the quorum Q_t. □

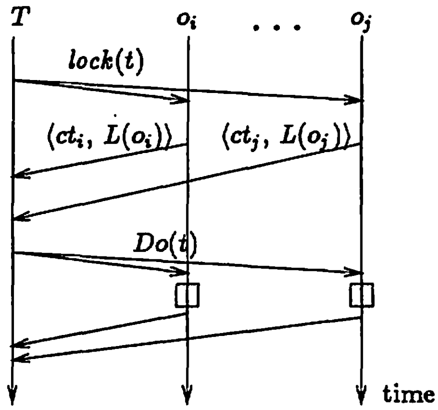


Figure 4: Type 2.

Next, let us consider the type 3 a new state of an object o obtained by performing a method t on a current state of o depends on the current state, i.e. $dtype(t) = Y$.

[Execution phase for type 3] [Figure 5] A state of an object o is changed by a method t depending on a current state, i.e. $dtype(t) = Y$.

1. Let L be a set $\{t' \mid Q_t \cap Q_{t'} \neq \phi, t' \in L(o_i)$, and $o_i \in Q_t\}$ of methods whose locks are held on replicas in the quorum Q_t and are compatible with the method t. $R_t = \{o_i \mid o_i \in Q_t \wedge ct_i \geq ct_j$ for every o_j in $Q_t\}$.
2. The method t is performed on every replica in the quorum Q_t.
3. For every replica o_i in the subset R_t, a collection L_i of methods where $L_i = L - L(o_i)$ are obtained. L_i shows methods which are not performed on the replica o_i and by which a state of o_i is changed, i.e. $stype = Y$. The methods in L_i are issued to the replica o_i in the subset R_t.
4. The methods in the set L_i are performed on each replica o_i. □

Here, every replica has the same state as the others in the quorum Q_t since every method which changes the state is performed. Each time a method in the set L_i is performed on a replica o_i, the version counter ct_i of the replica o_i is incremented by one. Every replica o_i in the quorum Q_t has the same version counter ct_i.

Let o_i and o_j be replicas of an object o. A replica o_i is referred to as *newer* than another

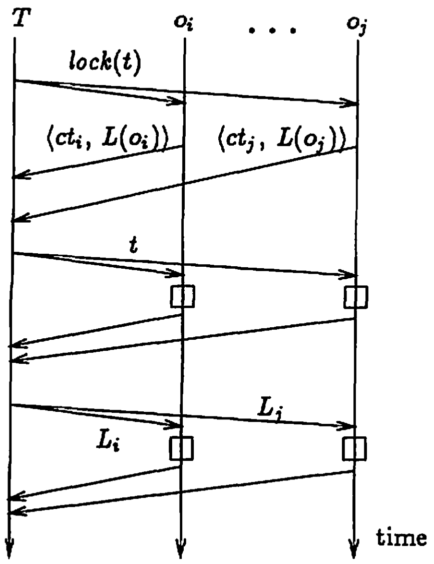


Figure 5: Type 3.

replica o_j if $ct_i \geq ct_j$. A replica o_i is newest in a cluster $C(o)$ iff there is no replica o_i such that o_j is newer than o_i.

It is straightforward for the following theorem to hold according to the protocol.

[Theorem] If a method t such that $otype(t) = Y$ is issued to an object o, the data is derived from the newest replica in a cluster $C(o)$ by the method t. □

According to the definition of the quorum, even if one of methods t and u changes an object o, $N_t + N_u \leq n$ for a total number n of the replicas. In traditional quroum-based protocols, $N_t + N_u > n$ is required to be held. Thus, we can reduce the number of replicas to be locked in the protocol.

5 Concluding Remarks

In object-based systems, objects are manipulated through methods which are implemented in procedures. In this paper, we discussed how replicas of objects are locked for abstract methods in the quorum-based scheme. Abstract methods are first classified with respect to whether or not data is derived, whether state is changed depending on a current state or independently of a current state. We defined the quorums for abstract methods based on the method types. Then, we discussed the quorum-based protocol to lock replicas and to perform the methods on the replicas. The protocol is composed of three phases, locking, decision, and execution phases depending on types of methods. By using the protocol, the number of replicas to be locked can be reduced.

Acknowledgment

This research is partially supported by Research Institute for Technology, Tokyo Denki University.

References

[1] Ahamad, M., Dasgupta, P., LeBlanc R., and Wilkes, C., "Fault Tolerant Computing in Object Based Distributed Operating Systems," *Proc. 6th IEEE SRDS*, 1987, pp. 115–125.

[2] Barrett, P. A., Hilborne, A. M., Bond, P. G., and Seaton, D. T., "The Delta-4 Extra Performance Architecture," *Proc. 20th Int'l Symp. on FTCS*, 1990, pp. 481–488.

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

[4] Bernstein, P. A., and Goodman, N., "The Failure and Recovery Problem for Replicated Databases," *Proc. 2nd ACM POCS*,

[5] Birman, K. P. and Joseph, T. A., "Reliable Communication in the Presence of Failures," *ACM TOCS*, Vol. 5, No. 1, pp 1987, pp. 47–76.

[6] Borg, A., Baumbach, J., and Glazer, S., "A Message System Supporting Fault Tolerance," *Proc. 9th ACM Symp. on Operating Sys. Principles*, 1983, . 27–39.

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

[8] Chevalier, P. -Y., "A Replicated Object Server for a Distributed Object-Oriented System," *Proc. IEEE SRDS*, 1992, pp.4-11.

[9] Date, C. J., "An Introduction to Database Systems," Addison Wesley, 1990.

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

[11] Gifford, D. K., "Weighted Voting for Replicated Data," *Proc. 7th ACM Symp. on Operating Systems Principles*, 1979, pp. 150-159.

[12] Hasegawa, K., Higaki, H., and Takizawa, M., "Object Replication Using Version Vector," *Proc. of the 6th IEEE Int'l Conf. on Parallel and Distributed Systems (ICPADS-98)*, 1998, pp. 147–154.

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

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

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

[16] Tanaka, K., Hasegawa, K., and Takizawa, M., "Quorum-Based Replication in Object-Based Systems," *Journal of Information Science and Engineering (JISE)*, Vol. 16, 2000, pp. 317–331.

[17] Tanaka, K. and Takizawa, M., "Quorum-Based Replication of Objects," *Proc. 3rd*

DEXA Int'l Workshop on Network-Based Information Systems (NBIS-3), 2000, pp. 33–37.