Flexible Group Communication for Dynamic Membership Changes

Takayuki Tachikawa, Hiroaki Higaki, and Makoto Takizawa
{tachi, hig, taki}@takilab.k.dendai.ac.jp
Dept. of Computers and Systems Engineering
Tokyo Denki University

In distributed systems, the group communication among multiple objects is required to do the cooperation. Kinds of group communication protocols have been discussed so far, which support the reliable and ordered delivery of messages. In distributed applications like teleconferences, the membership of the group is dynamically changed. For example, objects join and leave the group. If the group membership is changed, every member object in the group is required to agree on the membership. Furthermore, the messages sent by the member objects have to be causally delivered. In this paper, we would like to present a group communication protocol which provides the causally ordered delivery of messages while the membership is being changed.

メンバ構成の動的変化に対するやわらかいグループ通信

立川 敬行　桧垣 博章　滝沢 誠
東京電機大学理工学部経営工学科

分散型システムでは、複数のオブジェクトが協調動作を行うために，複数のオブジェクト間でのグループ通信が必要となる．これまでに，様々なグループ通信プロトコルが議論されており，特に，順序付けられた高信頼メッセージ配送が議論されている．電子会議のような分散型応用では，オブジェクトの加入，退会，オブジェクトの障害等により，グループのメンバ構成が変化する．グループのメンバ構成の変化に対して，グループの正しいメンバ構成についての合意を取ることと，グループの正しい構成オブジェクト間でメッセージを因果順に配送することが必要となる．本論文では，このためのグループ通信プロトコルを提案する．

1 Introduction

Distributed systems are composed of multiple computers connected by communication networks. In distributed applications like teleconferences and teleclassrooms [4], a group of multiple objects have to be cooperated. The group communication protocol is required to coordinate the cooperation of the objects in the group.

Current distributed systems are based on the client-server model. To support more reliable services, a group of multiple server replicas support the clients with the service even if some replicas are faulty. Here, the messages sent by the clients are multicast to the group of the server replicas. This type of communication is referred to as *multicast* and is discussed in many papers [2, 14].

While the multicast supports communication between a client and a group of server replicas, there is a requirement that a *group* of *autonomous* objects be established so that the objects communicate with each other in the group. This type of communication is adopted to teleconferences, parallel processing, and routing processing in internetworking. In the group communication, the following services have to be supported:

(G1) A message sent by the member object is received by one or multiple destination members in the group.

(G2) A member object in the group receives messages in the causal order [2].

In the group communications discussed by Takizawa, Tachikawa, and Nakamura [9,10,15,16], the membership of the group is fixed. That is, if the membership is changed, the group is closed and a new group is established again. In the teleconferences, some new member joins the conference and a member leaves the conference. Furthermore, some object may be faulty and may not be communicated due to the network partition.

If the membership of the group is changed, every member object has to reach agreement on the membership, i.e. what objects are included in the group. In addition, the messages sent by the member objects have to be causally delivered to every member objects in the group. By the group membership protocol, only and all the member objects make agreement on the membership of the group. In the papers [2,13], the membership protocols are discussed. Reiter [13] discusses a centralized membership protocol where one coordinator object coordinates the cooperation among the objects and the data transmission is stopped during the execution of the membership protocol. The protocol is robust to the Byzantine faults of the objects by enciphering the messages with digital signature. These protocols assume that the network is reliable and the faulty objects can be detected by the underlying system. In this paper, we would like to discuss how to support the services (G1) and (G2) without stopping the data transmission in the presence of the membership change. Furthermore, we would like to discuss how to support these services by using the distributed membership protocol, i.e. no controller object exists. This type of the group communication is referred to as "*flexible*."

In section 2, we would present a system model. In section 3, we discuss changes of the group. In section 4, we present the causal delivery. We discuss a membership protocol in sections 5 and 6.

2 System Model

A group G is composed of multiple objects O_1, ..., O_n ($n \geq 2$) interconnected by reliable high-speed networks [Figure 1]. Each object O_i is given to be a pair of data Δ_i and collection Π_i of operations for manipulating Δ_i. O_i has a role R_i in G, which is specified in terms of operations, i.e. $R_i \subseteq \Pi_i$. O_i can be manipulated through operations in R_i. The objects in G are cooperated.

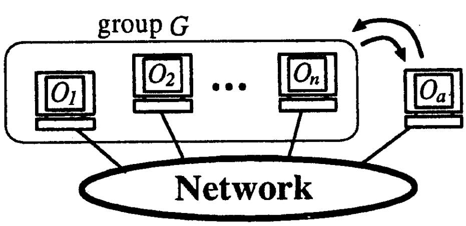

Figure 1: Group

In this paper, we make the following assumptions.

[Assumptions]

(1) There is a reliable, synchronous communication link between every two objects. That is, messages sent by each object are delivered to the destinations in the FIFO order. Messages are neither lost, contaminated, nor duplicated. In addition, the transmission delay is bounded to be δ time units.

(2) Each object communicates with each other through the communication network. That is, there is no shared memory among the objects.

(3) The objects may stop by fault. No other faults occur, i.e. no Byzantine fault [13].

(4) Network partitions may occur, e.g. due to the faults of routers. □

3 Changes in Group

3.1 Complete group

We have to discuss the following points if a group G is changed:

(1) who detects the membership change of G,

(2) how only and all the objects in G make agreement on the membership of G, and

(3) how the messages sent by the objects are causally delivered to the destinations in G while the membership of G is being changed.

While ISIS [2] assumes that the underlying system can detect the membership change, we assume that each object detects the membership change in G. For example, an object O_i detects that another O_j is faulty if O_i had not received any message in predetermined time units. Here, O_j is considered to leave G. O_i also detects that O_j would like to join G if O_i receives the request from O_j. Then, O_i informs the other member objects in G of the membership change. Here, the objects may have different views. That is, one object considers that O_j is a member of G but the other thinks not.

Each object O_i in the group G has a view $view_i(G)$ which denotes what objects O_i perceives are included in G. If G is not changed, every member object of G has the same view. If O_i is in G or would like to be in G, $O_i \in view_i(G)$. If $O_i \in view_i(G)$, O_i is referred to as ***participated*** in G. If $O_j \in view_i(G)$, O_j is referred to as ***recognized*** in G by O_i. If $O_j \in view_i(G)$ and $O_i \in view_j(G)$, O_i and O_j are referred to as ***agree*** with one another. $view_i(G)$ is changed if O_i finds the membership change of G. If O_i finds that an object O_k leaves G, O_k is removed from $view_i(G)$. Even if O_i finds O_k's leaving, another O_j may not find it. Thus, every pair of views $view_i(G)$ and $view_j(G)$ are not always identical. O_i and O_j have to agree on the membership of G. If O_i and O_j are participated in G and $view_i(G) \cap view_j(G) \neq \phi$, O_i and O_j are referred to as ***linked***. O_i and O_k are ***related*** if (1) O_i and O_j are linked and (2) O_j and O_k are related. Let $rel(O_i)$ be a set of objects which are related with O_i.

[Complete group] For every pair of objects O_i and O_j, a collection of objects $G = rel(O_i)$ is referred to as *complete* group if $O_j \in rel(O_i)$, $rel(O_i) = rel(O_j)$, and $view_i(G) = view_j(G)$. □

If a group G is incomplete, the objects in G reach no agreement on the membership of G. That is, some membership change occurs in G but no agreement on the membership is made yet in G.

[Example] Suppose that a group G is composed of three objects A, B, and C. Here, each object has the same view, i.e. $view_A(G) = view_B(G) = view_C(G) = \{ A, B, C \}$. Hence, G is complete [Figure 2]. In Figure 2, a directed edge $\alpha \rightarrow \beta$ denotes $\beta \in view_\alpha(G)$.

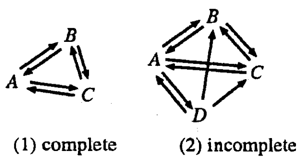

Figure 2: Complete group

Next, suppose that an object D joins G. Here, suppose that A regards D as a member of G, i.e. $view_A(G) = \{ A, B, C, D \}$. For example, A receives a join request from D and D knows that G includes A, B, and C. However, B and C do not yet agree that D is a member of G. Here, $\{ A, B, C, D \}$ is not complete since $view_A(G) \neq view_B(G) = view_C(G)$. Here, G is not complete. □.

3.2 Membership changes

The membership of the group G is changed if some member objects leave G, new objects join G, or member objects are faulty [Figure 3]. In this paper, we assume that an object sends *join* and *leaving* requests to G if the object would like to join and leave G, respectively. If the membership is changed, every object has to agree on the following points:

(1) what objects join/leave the group, and
(2) when the objects join/leave the group.

Here, let O denote a possible set of objects. The membership change is formalized to be a mapping *m-change*: $2^O \times O \rightarrow 2^O$. Here, suppose that a membership of a group G is composed of multiple objects $O_1, ..., O_n$, i.e. $G = \{ O_1, ..., O_n \}$ $(n \geq 2)$.

[Membership changes]

(1) An object O_{n+1} joins the group G, i.e. *m-change* $(\{ O_1, ..., O_n \}, O_{n+1}) = \{ O_1, ..., O_n, O_{n+1} \}$.
(2) An object O_i leaves G, i.e. *m-change* $(\{ O_1, ..., O_n \}, O_i) = \{ O_1, ..., O_{i-1}, O_{i+1}, ..., O_n \}$.
(3) A role R_i of O_i changes to R_i', i.e. $\langle R_i{:}O_i \rangle$ is changed to $\langle R_i'{:}O_i \rangle$. □

In this paper, we would like to discuss the membership changes of (1) and (2). The change of the role (3) is discussed in other papers.

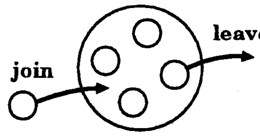

Figure 3: Membership change

4 Causally Ordered Delivery of Messages

Suppose that a group G is composed of multiple objects $O_1, ..., O_n$. Messages sent in G are required to be delivered in the causal order $\rightarrow$ [2].

[Causal order] A message m_1 *causally precedes* m_2 $(m_1 \rightarrow m_2)$ iff

(1) an object sends m_1 before m_2,
(2) an object sends m_2 after receiving m_1, or
(3) there exists a message m_3 such that $m_1 \rightarrow m_3 \rightarrow m_2$. □

Messages can be ordered by using the vector clock [8,11,12]. In the system of the vector clocks, the time domain is represented by a set of n-dimensional vector.

[Vector operations] For every pair of vectors $VC_1 = \langle VC_{11}, ..., VC_{1n} \rangle$ and $VC_2 = \langle VC_{21}, ..., VC_{2n} \rangle$, the following relation holds:

(1) $VC_1 = VC_2$ iff $VC_{1i} = VC_{2i}$ for $i = 1, ..., n$.
(2) $VC_1 < VC_2$ iff $VC_{1i} \leq VC_{2i}$ for $i = 1, ..., n$ and $VC_{1j} < VC_{2j}$ for some j.
(3) $max(VC_1, VC_2) = \langle VC_{31}, ..., VC_{3n} \rangle$. Here, $VC_{3i} = max(VC_{1i}, VC_{2i})$ for $i = 1, ..., n$. □

A vector time VC is given in a vector $\langle VC_1, ..., VC_n \rangle$ where each element VC_i represents an object O_i in a group $G = \langle O_1, ..., O_n \rangle$. The *scheme* of VC means which object in G each element VC_i represents. If the membership is changed, the scheme also has to be changed.

O_i has a variable $VC_i = \langle VC_{i1}, ..., VC_{in} \rangle$ denoting a vector time. VC_{ij} is initially 0 for $j = 1, ..., n$. Each message m sent by O_i carries a timestamp $m.VC = \langle m.VC_1, ..., m.VC_n \rangle$. O_i sends and receives messages by the following rule.

[Vector clock rule]

(1) Each time O_i sends a message m, $VC_{ii} := VC_{ii} + 1$; $m.VC := VC_i$;
(2) Each time O_i receives a message m from O_j, $VC_i := max(VC_i, m.VC)$; □

The following proposition [8, 12] on the vector clock holds.

[Proposition] For every pair of messages m_1 and m_2, $m_1 \rightarrow m_2$ iff $m_1.VC < m_2.VC$. □

The objects cannot detect message loss by using the vector clock [8, 10]. Nakamura and Takizawa [10] present a protocol by which message loss can be detected and messages can be causally delivered by using the sequence numbers of the messages.

5 Membership Management

In this section, we would like to present a method for detecting the membership change and making agreement on the membership changed.

5.1 Timestamp scheme

We would like to discuss how to manage the membership of a group G. The membership of G is changed if a new object joins G, some object leaves G, or some object is detected to be faulty in G. If an object O_j would like to leave or join G, O_j first notifies of it to some object O_i in G. If some O_j is faulty in G, an object O_i detects the fault of O_j if O_i had received no message from O_j in predetermined time units, i.e. timeout. Then, O_i initiates the membership protocol to make agreement on the membership.

Here, let O be a set of possible objects. For a group G, let G_k be a membership of G, i.e. $G_k \subseteq O$. The membership G_k of G is changed to G_{k+1} ($\subseteq$ O) if the membership is changed. If G_k is changed to G_{k+1}, all the objects in G_{k+1} have to agree on G_{k+1}. That is, if every object O_i in G_{k+1} has the same $view_i(G)$, every object makes agreement on the membership G_{k+1}. Here, G_k is referred to as the kth version of G. The scheme of the vector clock VC_i denotes the view $view_i(G)$ of O_i. If O_i detects the membership change, the vector clock scheme of VC_i is updated in O_i so that the new scheme represents the new membership. The dimension of the vector clock is changed according to the update of the vector clock scheme. For example, if a new object O_{n+1} joins G, VC_i of O_i is updated from $\langle VC_{i1}, ..., VC_{in} \rangle$ to an (n + 1)- dimension vector $\langle VC_{i1}, ..., VC_{in}, VC_{i(n+1)} \rangle$. If the scheme of the vector clock is changed, the version of the vector clock is said to be changed. Each version is identified by the version number. Each object O_i has a variable ver_i which denotes the version number of VC_i. ver_i is initially 0. ver_i is updated by the following procedure.

[Update of version number] O_i receives a membership change request m from O_j.

(1) O_i increments the version number by one, i.e. $ver_i := ver_i + 1$.

(2) The vector clock scheme of VC_i is updated so as to denote the new membership notified by m. □

The version number of VC_i is carried back in a field $m.ver$ of a message m. The messages with the same version numbers can be causally ordered according to the proposition.

Every object has to be synchronize to update the version number. We take the distributed approach to synchronize the objects while Reiter [13] takes the centralize approach. In the distributed approach, there is no coordinator.

5.2 Detection of changes

The membership of the group G is changed if the following events occur in G:

(1) an object O_{n+1} joins G.

(2) an object O_j leaves G.

(3) an object O_j is faulty or cannot communicate with objects in G due to the network partition.

If O_{n+1} would like to join G, O_{n+1} sends a *join* request to one object, say O_i in G. Another object which would like to join G may send the join request to an object different from O_i. Similarly, if O_j would like to leave G, O_j sends a *leaving* request to one object, say O_i in G. The faulty object O_j is detected by an object, say O_i if O_i had not received one message from O_j for some predetermined time units, say 2δ. Then, O_i initiates the membership procedure presented in the succeeding section.

5.3 Membership procedure

Each object O_i has two kinds of variables, L_i and J_i. L_i denotes a set of objects which are detected to leave G, and J_i denotes a set of objects which are detected to join G. Initially, $L_i = J_i = \phi$ and O_i is in a *normal* state. While the membership of G is not changed, $L_i = J_i = \phi$. If O_i detects O_j's joining and leaving G, O_j is added to L_i and J_i, respectively. $G - L_i \cup J_i$ denotes a view $view_i(G)$ of O_i in G.

[Membership procedure]

(1) If L_i or J_i is changed, i.e., O_i finds the membership change, O_i sends a membership message m with L_i and J_i to all objects in $G \cup J_i$. O_i is in an *updating* state.

(2) On receipt of the membership message m with L_i and J_i from O_i, O_j manipulates L_j and J_j as $L_j := L_j \cup L_i$ and $J_j := J_j \cup J_i$. O_j is in an updating state.

(3) If L_j and J_j are changed, O_j sends the membership message with L_i and J_j to all the objects.

(4) If O_k receives the membership message with L_h and J_h from every object O_h in $G - L_k \cup J_k$, and $L_h = L_h$ and $J_h = J_h$, then O_k updates the membership of G to $G - L_k \cup J_k$. The version number ver_k is incremented by one. O_k leaves the updating state and is in a normal state. $L_k := J_k := \phi$. □

In an updating state, $L_i \neq \phi$ or $J_i \neq \phi$ in O_i. If O_i receives the membership change request like join and leaving or O_i finds the fault of another object, O_i stores the events in the log while O_i is in the updating state. On transiting to the normal state, O_i updates L_i and J_i by using the events in the log. Then, O_i initiates the membership procedure again.

It is noted that each object O_i can send normal messages in the updating state. That is, O_i does not stop the data transmission while the membership procedure is being executed.

Figure 4 shows an example of a group $G = \{ O_1, ..., O_5 \}$. O_3 would like to leave G and O_6 would like to join G. O_3 sends a leaving request r_1 to O_1. On receipt of r_1, $L_1 = \{O_3\}$ and $J_1 = \phi$. O_1 sends the membership message m_1 with L_1 and J_1 to all the objects, i.e. O_1, O_2, O_4, and O_5. O_6 sends a join request r_2 to O_5. On receipt of r_2, $L_5 = \phi$ and $J_5 = \phi$, and O_5 sends the membership message m_2 with L_5 and J_5 to all the objects. O_2 receives m_1 and m_2. Here, $L_2 = L_1 \cup L_5 = \{ O_6 \}$ and $J_2 = J_1 \cup J_5 = \{ O_3 \}$. Since L_2 and J_2 are changed, O_2 sends the membership message m_3 with L_2 and J_2 to all the objects in $G = G - L_2 \cup J_2 = \{ O_1, O_2, O_4, O_5, O_6 \}$. On receipt of m_3, L_6 and J_6 gets $\{ O_3 \}$ and $\{ O_6 \}$ in O_6, respectively and O_6 sends the membership message to all the objects. Here, every object in $\{ O_1, O_2, O_4, O_5, O_6 \}$ has the same view.

5.4 Fault in membership change

Next, we would like to consider a case that an object is being faulty in G when the objects are in an updating state. Let us consider a case that O_3 leaves the group $G = \{ O_1, O_2, O_3, O_4, O_5 \}$ in Figure 4. First, suppose that the initiatior O_1 of the membership procedure is faulty. There are the following cases:

(1) O_1 faults before sending the membership message m_1.
(2) O_1 faults after sending m_1.

If the fault (1) occurs, the other objects do not receive m_1. Hence, the objects detect the fault of O_1 by timeout. Then, one object, say O_2 initiates the membership procedure.

If the fault (2) occurs, the other objects detect the fault of O_1 by timeout. After the objects O_2, O_4, and O_5 agree on the membership $\{ O_1, O_2, O_4, O_5 \}$, the membership procedure is initiated to make agreement on $\{ O_2, O_4, O_5 \}$.

Next, let us consider the fault of the other object than O_1, say O_2. Suppose that O_2 faults before sending the membership message. The other object, say O_4 detects the fault of O_2 because O_4 does not receive the membership message. After receiving all the membership massages from O_1, O_4, O_5, the membership procedure is initiated to exclude O_2 from the membership.

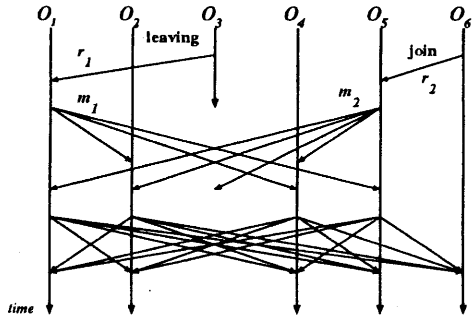

Figure 4: Membership change

[Theorem] For every pair of objects O_i and O_j in a membership of a group G, O_i and O_j have the same version number if the membership procedure terminates and O_i and O_j are included in the new membership of G. □

6 Delivery of Messages

The dimension of the vector clock is changed according to the membership change of the group G. Hence, each object may receive messages with different version numbers. Suppose that an object O_i receives two messages m_1 and m_2. If m_1 and m_2 have different version numbers, i.e. $m_1.ver \neq m_2,ver$, the vector times $m_1.VC$ and $m_2.VC$ cannot be compared because they have different dimensions of the vector clocks. Hence, O_i cannot causally order m_1 and m_2. If $m_1.VC$ and $m_2.VC$ have the same scheme, i.e. $m_1.ver = m_2.ver$, O_i can decide how m_1 and m_2 are causally preceded by comparing $m_1.VC$ and $m_2.VC$. m_1 and m_2 are ordered by the following rule.

[Ordering (O) rule] For every pair of messages m_1 and m_2, m_1 *precedes* m_2 if the following condition holds:

(1) if $m_1.ver = m_2.ver$, $m_1.VC \leq m_2.VC$,
(2) otherwise $m_1.ver \leq m_2.ver$. □

[Theorem] m_1 causally precedes m_2 if m_1 precedes m_2 by the O rule. □

We would like to present a protocol to causally deliver messages while the membership of G is being changed. Each object O_i has a variable ver_i denoting the current version number. Suppose that O_i receives a message m from O_j. There are following three cases:

(1) $m.ver > ver_i$.
(2) $m.ver < ver_i$.
(3) $m.ver = ver_i$.

We would like to consider the first case (1) $m.ver > ver_i$. This means that O_j sends m to

O_i after the version of the vector scheme is updated while O_i's version is not updated yet. O_i stores m in the buffer. m is stayed in the buffer until ver_i is updated.

The second case (2) $m.ver < ver_i$ means that O_j sends m to O_i before updating the vector clock while O_i has updated the vector clock. Thus, O_i may receive messages with older version numbers than O_i. Here, O_i receives messages from the objects in the new membership. These messages have the same version number as ver_i. Suppose that O_i receives a message m_j from O_j where $m_j.ver = ver_i$. However, O_i does not deliver m_j by the O rule because O_i might still receive messages whose version number is smaller than ver_i, i.e. messages sent in the old version of G. Here, O_i stores m_j in the buffer. If the following condition holds, the messages stored in the buffer are causally delivered according to the O rule.

[Change of vector clock scheme] O_i receives a message with the new vector clock scheme from every object in G. □

Finally, we would like to consider the case (3) $m.ver = ver_i$. In this case, O_i delivers m in the causal order by the causality rule.

[Theorem] By the membership protocol, messages are causally delivered without stopping the data transmission even if the membership of the group is changed. □

7 Concluding Remarks

In this paper, we have presented the group communication protocol for maintaining the membership of the group G and supporting the causally ordered delivery of messages while the membership is being changed. We have adopted the distributed protocol where there is no centralized controller while ISIS takes the decentralized approach. By using the protocol, the objects can reach agreement on the membership without stopping data transmission. The protocol can apply to distributed applications where multiple autonomous objects are cooperated with each other like teleconferences.

References

[1] Agarwal, D. A., Moser, L. E., Melliar-smith, P. M., and Budhia, R. K., "A Reliable Ordered Delivery Protocol for Interconnected Local-Area Networks," *Proc. of IEEE ICNP-95*, 1995, pp. 365–374.

[2] Birman, K., Schiper, A., and Stephenson, P., "Lightweight Causal and Atomic Group Multicast," *ACM Transcations on Computer Systems*, Vol. 9, No. 3, 1991, pp. 272–314.

[3] Comer, D., "Internetworking with TCP/IP: Principles, Protocols and Architectures," *Prentice Hall, Englewood Cliffs, NJ*, 1988.

[4] Ellis, C. A., Gibbs, S. J., and Reln, G. L., "Groupware Some Issues And Experiences," *Comm. of the ACM*, Vol. 34, No. 1, 1991, pp. 39–58.

[5] Higaki, H., "Group Communications Algorithm for Dynamically Updating in Distributed Systems," *Proc. of the IEEE ICPADS*, 1994, pp. 56–62.

[6] Kaashoek, M. F. and Tanenbaum, A. S., "Group Communication in the Amoeba Distributed Operating System," *Proc. of IEEE ICDCS-11*, 1991, pp. 222–230.

[7] Lamport, L., "Time, Clocks, and the Ordering of the Events in a Distributed System," *Comm. of the ACM*, Vol. 21, No. 7, 1978, pp. 558–565.

[8] Mattern, F., "Virtual Time and Global States of Distributed systems," *Proc. of Parallel and Distributed Algorithms Conf.*, 1988, pp. 215–226.

[9] Nakamura, A. and Takizawa, M., "Reliable Broadcast Protocol for Selectively Ordering PDUs," *Proc. of IEEE ICDCS-11*, 1991,

[10] Nakamura, A. and Takizawa, M., "Causally Ordering Broadcast Protocol," *Proc. of IEEE ICDCS-14*, 1994, pp. 48–55.

[11] Raynal, M., "About logical clocks for distributed systems," *ACM Operating Systems Review*, Vol. 26, No. 1, 1992, pp. 41–48.

[12] Raynal, M. and Singhal, M., "Logical time: capturing causality in distributed systems," *IEEE Computer*, Vol. 29, No. 2, 1996, pp. 49–56.

[13] Reiter, M. K., "A Secure Group Membership Protocol," *IEEE Trans. on Software Engineering*, Vol.22, No.1, 1996, pp. 31–42.

[14] Speirs, N. A. and Barretti, P. A., "Using Passive Replicates in Delta-4 to Provide Dependable Distributed Computing," *Proc. of IEEE FTCS19*, 1989, pp. 184–190.

[15] Tachikawa, T. and Takizawa, M., "Selective Total-Ordering Group Communication on Single High-Speed Channel," *Proc. of IEEE ICNP-94*, 1994, pp. 212–219.

[16] Takizawa, M. and Nakamura, A., "Partially Ordering Broadcast (PO) Protocol," *Proc. of IEEE INFOCOM'90*, 1990, pp. 357–364.