マルチメディア通信と分散処理 71−22
（1995. 7. 14）

フォールトトレラントグループ通信

島 健司 滝沢 誠
東京電機大学理工学部経営工学科
E-mail {sima, taki}@takilab.k.dendai.ac.jp

分散型システムは、通信網で接続された複数のプロセスが協調動作を行ない、ある目的を達成するための計算を行なう。このとき、グループ内のプロセスに、原子的で因果順のメッセージ配送を行なう必要がある。プロセスの種々の障害に対しても、計算を中断することなく、継続し、結果を出せる必要がある。本論文では、プロセスの停止障害だけでなくビザンチン障害も考慮して、複数のプロセス間でのグループ通信を行なうために、プロセスを複数のレプリカに多重化する方法ついて論じる。

Fault-Tolerant Intra-Group Communication

Kenji Shima and Makoto Takizawa
Tokyo Denki University

In distributed applications, a group of processes have to be cooperated. The intra-group communication supports the atomic and causally ordered delivery of messages with the processes in the group. Each process in the group is replicated into a collection of multiple replicas named a clusters. In this paper, we would like to discuss a fault-tolerant group communication which supports the atomic and ordered delivery of messages among the clusters in the group in the presence of Byzantine faults of the replicas.

1 Introduction

In distributed applications like teleconferences and telemedicines [6], a group of multiple application processes have to be communicated with one another. Multiple *system* processes have to support the application processes with the *atomic* and *ordered* delivery of messages by using the high-speed network. By using the group communication service supported by the system processes, the application processes can send atomically messages to the others in the group in some delivery order like the *causal order* [3,16]. The distributed systems suffer from the faults of system processes and network. [?, 14, 15, 20] discuss how to detect and recover from the message loss, i.e. *omission fault* in the network, where the group is aborted if some system process faults. [2, 13] discuss how to manage the membership of the group in the presence of process fail-stop. In this paper, we assume that the underlying network is reliable, i.e. the system process can deliver messages to all of the destinations without message loss. From here, let *processes* mean system processes. Even if the processes in the group fault, it is required to support the group of application processes with the atomic and ordered delivery of messages.

There are two kinds of group communications. In one case, processes send messages to a group or groups of processes. For example, clients send messages of data manipulation requests to a group of replicated database servers. It is the *inter-group* communication discussed in [2, 3, 10, 21]. In another case, only processes in the group communicate with one another. It is the *intra-group* communication [?, 14–16, 20]. Here, after a collection of application processes has established a group, the processes send messages to the other processes in the group. In this paper, we discuss the fault-tolerant intra-group communication, where the application processes can communicate with each other in the group in the presence of the process faults. In this paper, we make no assumption on the process faults, i.e. the process may suffer from Byzantine fault [12]. In order to support the fault-tolerant group communication, each process is replicated into a collection of *replicas*, which is named a *cluster*. Even if some replica faults, the process is considered to be operational if more than half of the replicas are operational in the cluster, i.e. the application process can still take the group communication service through the operational replicas. There are three kinds of replication strategies, i.e. *passive*, *active*, and *semi-active* replications [4, 17, 21]. In the active replication [17], all the replicas do the same computation by receiving and sending the same messages. In the passive [4] and semi-active [21] replications, only one replica named *coordinator* does the computation of the process. In the active replication, the total processing time of the replicas and total communication time are larger than the others, because all the replicas compute and communicate. However, the semi-active and passive repli-

cations can not tolerate the Byzantine fault of the coordinator.

The *fault-tolerant* group is composed of multiple *clusters* where each cluster is a collection of replicas of a process. In order for a process p_i to send a message m to p_j, the replicas in a cluster of p_i send messages m to the replicas in a cluster of p_j in the fault-tolerant group, while one process sends one message m to p_j in the conventional group. In order to detect the faulty replicas, each replica has to receive messages from multiple replicas. It is problem how to reduce the number of messages transmitted between the clusters. Furthermore, the replicas in the cluster may receive messages in different orders. In this paper, we discuss how the clusters communicate with each other in the group and how the replicas support the processes with the atomic and ordered delivery of messages.

In section 2, we discuss the replication schemes. In section 3, we present the properties of the group communication. In section 4, we present a fault-tolerant group. In section 5, we discuss the inter-cluster communication in the fault-tolerant group. In section 6, we discuss how to support the causally ordered delivery of messages.

2 Replication Schemes

First, we would like to consider how to replicate a process p_i into multiple replicas $p_{i1}, \ldots, p_{il_i}$ ($l_i \geq 1$). Three kinds of schemes for replicating p_i is discussed in [4, 17, 21]:

(1) *active* replication,

(2) *passive* replication, and

(3) *semi-active* replication.

In the active replication [17], every replica p_{ij} does the same computation by receiving and sending the same messages. Even if some replica of p_i faults, the computation of p_i can be continued without stopping as long as at least one replica is operational. The computation of p_i is required to be deterministic because every replica has to do the same computation. By comparing the messages with the others sent by the replicas, the faulty replicas can be detected if the replicas send messages different from ones which the majority of the replicas have sent.

In the passive replication [4], there is one replica named a *coordinator*, say p_{i1}. The other replicas $p_{i2}, \ldots, p_{il_i}$ are named *participants*. p_{i1} receives and sends messages and computes while no participant computes. p_{i1} takes the checkpoint, e.g. periodically, where p_{i1} saves the local state in the local stable storage named *log* l_{i1}. Here, p_{i1} sends the local state ls_{i1} taken at the checkpoint ck_{i1} to all the participants. On receipt of ls_{i1} from p_{i1}, every participant p_{ij} saves ls_{i1} into the stable storage l_{ij} at the checkpoint and changes the local state to ls_{i1}. p_{ij} starts the computation of p_i from the checkpoint ck_{ij} taken most recently if the coordinator p_{i1} faults. Then, a new coordinator is selected among the operational participants and restarts the computation from the checkpoint. This means that it takes some time for the replicas p_{ij} to roll back to ck_{ij} and restart from ck_{ij}. The passive replication can be adopted to the non-deterministic processes because only one replica, i.e. coordinator does the computation and the others catch up with it on receiving the checkpoint from the coordinator.

In the semi-active replication [21], all the replicas receive the same messages and do the same computation, but the non-coordinator replicas send no message while only the coordinator p_{i1} sends the messages. Like the passive replication, one coordinator p_{i1} takes checkpoint and sends the local state ls_{i1} taken at the checkpoint to all the participants. On receipt of the local state ls_{i1}, each participant p_{ij} changes the local state to ls_{i1}. Even if the current state of p_{ij} is different from ls_{i1}, p_{ij} changes the state to ls_{i1}. Here, the processes may be non-deterministic .

The active replication implies larger processing and communication overhead than the passive and semi-active ones because all the replicas do the same computation by sending and receiving the same messages. The passive replication has less overhead than active and semi-active ones because only coordinator does the computation and sends and receives messages. However, it takes time to recover from the fault of the coordinator because one coordinator has to be selected and the computation of the process has to be restarted from the checkpoint. Since there is one coordinator, if the coordinator suffers from the Byzantine fault, it cannot be detected.

3 Group Communication

A distributed system is composed of multiple processors interconnected by communication networks. A distributed application is realized by the cooperation of $n(\geq 2)$ processes $p_1, \ldots, p_n$ each of which is computed in one processor. A collection of $p_1, \ldots, p_n$ is referred to as *group* G, written as $G = < p_1, \ldots, p_n >$. Each process p_i is modeled as a finit state machine [17], where the state is transited if an event occurs. Hence, the computation of p_i is considered to be a sequence of events occurring in p_i. There are two kinds of events, i.e. *internal* and *communication* events. There are two kinds of communication events, i.e. *sending* and *receipt* events of messages. Here, let $r_i(m)$ and $s_i(m)$ denote receipt and sending events of a message m by p_i, respectively. It is important to consider the ordered relation on the events occurring in the distribution system. Lamport [11] defines the *happened-before* relation $\rightarrow$ on the events.

[Definition] For every pair of events e_1 and e_2, e_1 *precedes* e_2 ($e_1 \rightarrow e_2$) iff

(1) e_1 happens before e_2 in p_i,

(2) for some (not necessarily different) p_i and p_j,

there exists some message m such that $e_1 = s_i(m)$ and $e_2 = r_j(m)$, or

(3) for some event e_3, $e_1 \rightarrow e_3$ and $e_3 \rightarrow e_2$. □

Then, a causal relation among the messages is defined [3].

[Definition] For every pair of messages m_1 and m_2, m_1 *causally precedes* m_2 ($m_1 \prec m_2$) iff $s_i(m_1) \rightarrow s_j(m_2)$. □

The causal precedence relation $\prec$ is transitive but not symmetric. The relation $\prec$ among messages reflects the happened-before relation $\rightarrow$ among the sending events of the messages. Intuitively speaking, $m_1 \prec m_2$ means that m_1 is sent before m_2 in the distributed system. m_1 and m_2 are referred to as *causally-coincident* ($m_1 \parallel m_2$) if neither $m_1 \prec m_2$ nor $m_2 \prec m_1$. $m_1 \preceq m_2$ iff $m_1 \prec m_2$ or $m_1 \parallel m_2$.

If the communication system satisfies the following rule, it supports the causally ordered delivery of messages.

[Causally ordered (CO) delivery] For every pair of messages m_1 and m_2, deliver m_1 before m_2 if $m_1 \prec m_2$. □

In Figure 1, there are three processes p_j, p_h, and p_k. After sending m_0 to p_k, p_j sends a message m_1 to p_h and p_k. p_h sends m_2 to p_k after receiving m_1. Here, $m_0 \prec m_1 \prec m_2$. Hence, if p_k receives m_0, m_1, and m_2 in this sequence the messages are causally delivered. If p_k receives m_1 after m_2 as denoted by the dotted line, the messages are not causally delivered.

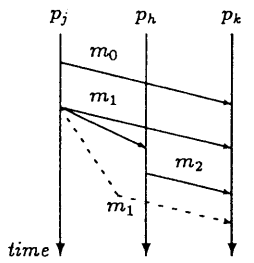

Figure 1: Causally ordered delivery

A process p_i sends a message m to the processes in the group G. If the communication system satisfies the following rule, it supports the atomic delivery of messages

[Atomic (A) delivery] Deliver a message m only if all the destination processes of m receive m. □

If at least one process does not receive m, m is not delivered. That is, all the processes either receive m or none of them.

4 Fault-Tolerant Group

The fault-tolerant group for a group $G = < p_1, \ldots, p_n >$ is composed of n clusters $c(p_1), \ldots, c(p_n)$. Each cluster $c(p_i)$ is a collection of replicas $p_{i1}, \ldots, p_{il_i}$ ($l_i \geq 2$) of p_i [Figure 2]. All replicas

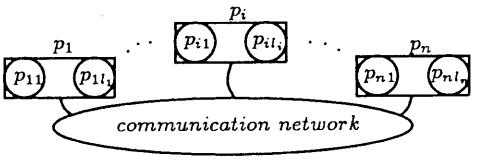

Figure 2: Fault-tolerant group

in $c(p_i)$ behave identically, i.e. actively replicated where each replica is the deterministic finite state machine which receives the same messages, does the same computation, and sends the same messages.

We make the following assumptions on the network.

(1) If a process p_j sends messages m_1 before m_2 to p_i, p_i receives m_1 before m_2.

(2) The network is *synchronous* [7], i.e. the propagation delay is bounded.

(3) If p_j sends a message m to p_i, p_i receives exactly one message m.

(4) If p_i receives a message m, m is sent by one process.

These assumptions mean that the network is reliable.

We make the following assumptions on the process fault:

(1) the processes are reliable.

(2) the processors may fault, and

(3) the faults occur independently.

Due to the processor fault, the process on the faulty processor may behave incorrectly, i.e. does not satisfy the specification of the process. That is, the process is viewed to suffer from the Byzantine fault. The process which does not satisfy the specification is referred to as *fault*. The replicas are interconnected by the communication network. We assume that at most f_i ($\leq l_i$) replicas fault at the same time in each cluster $c(p_i)$. The replicas are allocated to different processors to realize the independent fault assumption.

5 Inter-Cluster Communication

In the fault-tolerant group communication, a cluster $c(p_i) = \{p_{i1}, \ldots, p_{il_i}\}$ has to send a message m to $c(p_j) = \{p_{j1}, \ldots, p_{jl_j}\}$ in order for p_i to deliver m to p_j. Each replica p_{jh} receives messages for multiple replicas in $c(p_i)$. Some messages are sent by faulty replicas. Hence, the replicas in $c(p_i)$ have to detect messages sent by the correct replicas in $c(p_i)$. $c(p_j)$ has to receive messages from more than $2f_i + 1$ replicas in $c(p_i)$. If $c(p_j)$ receives the same message from at least $f_i + 1$ replicas in $c(p_i)$, $c(p_j)$ accept the message. In addition, $c(p_i)$ considers that the replicas which have sent the messages different from the message

sent by $f_i + 1$ replicas. If only fail-stop [18] fault occurs, $c(p_j)$ has to receive at least one message from $c(p_i)$.

We would like to discuss how the clusters in the group communicate with each other. Suppose that a process p_i sends a message m to p_j. There are two ways for the cluster $c(p_i)$ to send m to $c(p_j)$:

(1) each p_{ik} in $c(p_i)$ sends m to all replicas $p_{j1}, \ldots, p_{jl_j}$ in $c(p_j)$,

(2) each p_{ik} sends m to a subset $I_j(p_{ik})$ of $c(p_j)$ for $k = 1, \ldots, l_i$.

In the first method, each p_{ik} sends m to l_j replicas in $c(p_j)$ $(j \neq i)$. Hence, p_{ik} sends totally $l_1 + \cdots + l_{i-1} + l_{i+1} + \cdots + l_n$ messages to $n-1$ cluster $c(p_1), \ldots, c(p_{i-1}), c(p_{i+1}), \ldots, c(p_n)$ and each p_{jh} receives $l_1 + \cdots + l_{j-1} + l_{j+1} + \cdots + l_n$ messages [Figure ??] if the one-to-one network is used. In order to deliver m to $n-1$ clusters $c(p_1), \ldots, c(p_{j-1}), c(p_{j+1}), \ldots, c(p_n)$, the cluster $c(p_i)$ sends totally $l_i \cdot (l_1 + \cdots + l_{i-1} + l_{i+1} + \cdots + l_n)$ messages. This method is named a *broadcast* distribution one.

If the broadcast network is used, l_i messages are transmitted if each replica in $c(p_i)$ sends one message. Since each p_{jh} needs to receive at most $2f_i + 1$ messages, only $2f_i + 1$ $(\leq l_i)$ replicas can broadcast messages. Here, totally $(2f_i + 1)$ messages are transmitted.

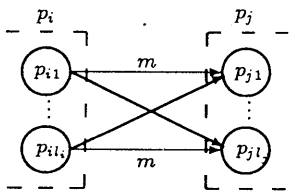

Figure 3: Inter-cluster communication

Another way is that each replica p_{ik} sends the message m to not all the replicas in $c(p_j)$, but only a subset $I_j(p_{ik}) \subseteq c(p_j)$. Each replica p_{jh} has to receive at least $2f_i + 1$ messages from $c(p_i)$ and does not need to receive more than $2f_i + 1$ messages. Hence, the following constraints have to be satisfied.

(1) $| \{ p_{ik} \mid p_{jh} \in I_j(p_{ik}) \} | \geq 2f_i + 1$ for every p_{jh}, and

(2) $I_j(p_{i1}) \cup \cdots \cup I_j(p_{il_i}) = c(p_j)$.

The total number of messages transmitted is $| I_j(p_{i1}) | + \cdots + | I_j(p_{il_i}) |$. If each p_{ik} sends m to $(2f_i + 1)l_j / l_i$ replicas of p_j, the minimum number $(2f_i + 1)l_j$ of messages are transmitted. Exactly saying, $l_i - [(2f_i + 1)l_j \ \textit{modulo}\ l_i]$ replicas send m to $\lceil (2f_i + 1)\ l_j\ /\ l_i \rceil$ replicas and $(2f_i + 1)l_j$ *modulo* l_i replicas send m to $\lfloor (2f_i + 1)l_j / l_i \rfloor$ replicas if $(2f_i + 1)l_j$ *modulo* $l_i \neq 0$. This method for distributing m from p_i to the cluster $c(p_j)$ is a *selective* broadcast distribution one.

Figure 4 shows that four replicas of p_i $(l_i = 4)$ send m to five replicas of p_i $(l_j = 5)$, where $f_i = 1$. p_{i1} sends three messages and p_{i2}, p_{i3}, and p_{i4} send four messages. Each p_{jh} receives three messages from p_i. Totally $(2f_i + 1)l_j = 3 \cdot 5 = 15$ messages are transmitted. If the broadcast distribution method is used, $4 \cdot 5 = 20$ messages are transmitted. If the broadcast network is used, only 3 replicas among four can send messages, because each p_{jh} receives three messages.

The replicas which receive messages from other clusters and send message to other clusters are referred to as *input* and *output* replicas, respectively. In Figure 3 and 4, all the replicas are input and output ones.

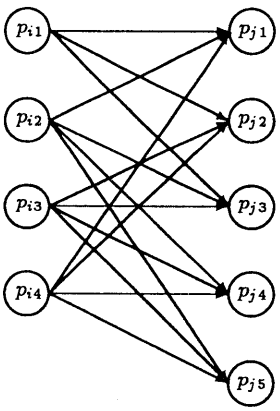

Figure 4: Selective broadcast distribution

6 Causally Ordered Delivery

In the group communication, the messages sent in the group have to be causally delivered to the application processes. In the fault-tolerant group communication, each processes p_i is replicated into a cluster of replicas, i.e. $c(p_i) = \{p_{i1}, \ldots, p_{il_i}\}$ $(i = 1. \ldots, n)$. The replicas send messages to the replicas in other clusters and receive messages from multiple replicas in other clusters, while each process receives one message from other processes in the conventional group. Here, some replica may not receive the messages and may receive the messages in different orders in each cluster.

In the fault-tolerant group communication, the communication system has to deliver messages to all the destination processes in an atomic manner. First, we would like to consider the atomic delivery of messages in the fault-tolerant group. Here, suppose that a process p_j sends a message m to p_i. That is, the replicas in $c(p_j)$ send m to the replicas in $c(p_i)$. If a message m arrives at p_{ik} from al least $f_i + 1$ replicas in $c(p_j)$, p_{ik} receives m.

[Cluster (C) delivery] The communication system delivers m to p_i if more than f_i replicas in the cluster $c(p_i)$ receive m. □

In the Byzantine fault, p_i receives m if at least $f_i + 1$ replicas receive m.

[Atomic replica (AR) delivery] The communication system delivers m only if it delivers m to all the destination processes of m. □

As discussed in the preceding section, each cluster can surely send a message to other clusters in the presence of the replica faults.

Next, the communication system has to deliver messages to the destinations processes in some order. The replicas in a cluster $c(p_i)$ may receive messages in different orders due to the communication delay. Furthermore, they may deliver the messages to p_i in different orders even if they accept the messages in the same order due to the Byzantine fault.

In this paper, we assume that each message has a unique identifier named a *sequence number sn* given by a process and the replicas cannot change the identifier. Here, let $m.sn$ be the sequence number sn of a message m. For every pair of messages m_1 and m_2 sent by p_i, $m_1.sn < m_2.sn$ iff m_1 is sent before m_2. Hence, the communication system can deliver messages sent by one process in the sending order.

[Cluster ordered (CO) delivery] For every pair of messages m_1 and m_2 sent by different clusters, the communication system delivers m_1 before m_2 to p_i if more than f_i replicas in $c(p_i)$ deliver m_1 before m_2. □

If $c(p_i)$ includes at least $2f_i + 1$ replicas, the messages can be delivered to p_i in some well-defined order.

[Fault-tolerant ordered (FTO) delivery] For every pair of messages m_1 and m_2, the communication system delivers m_1 before m_2 if m_1 is delivered before m_2 in all the clusters. □

That, the messages are delivered to all the processes in the same order. The group communication system has to support the causally ordered delivery of messages for the processes in the group. First, we would like to discuss the precedence relation among the messages. For two messages m_1 and m_2, m_1 is referred to *precede* m_2 at the replica level ($m_1 \prec m_2$) if $s_{ik}(m_1) \rightarrow s_{jh}(m_2)$.

As presented in [16, 19], each message m sent by p_{ik} carries the confirmation field ack_{jh} which denotes the sequence number of a message which p_{ik} expects to receive next from p_{jh} ($h = 1, \ldots, l_i$, $j = 1, \ldots, n_j$). The messages can be causally ordered as follows [16].

[Replica causally oredered (RCO)] For each pair of messages m_1 and m_2, $m_1 \prec m_2$ if

(1) $m_1.sn < m_2.sn$ if m_1 and m_2 are sent by the same cluster, and

(2) $m_1.sn < m_2.ack_{ik}$ if m_1 is sent by p_{ik} and m_2 is sent by the other replica. □

Here, it is noted that the faulty replica can change the confirmation fields in the messages while it cannot change the sequence numbers.

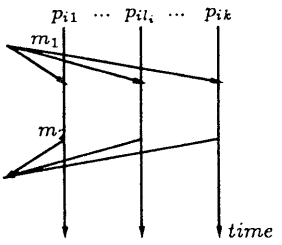

Figure 5: Precedence

This means that the faulty replica may send the incorrect precedence information to other replica.

[Message ordering] On receipt of messages m_1 and m_2, each replica p_{ij} can decide whether $m_1 \prec m_2$ or not. □

Suppose that all the replicas in $c(p_i)$ send a message m_2 after receiving m_1. On receipt of m_2 from p_{ik}, the replica p_{jh} knows that $m_1 \prec m_2$ in p_{ik} (written as $m_1 \prec_{ik} m_2$).

[Replica perception] The replica p_{jh} perceivers $m_1 \prec m_2$ if $| \{ p_{ik} \mid m_1 \prec_{ik} m_2 \} | \geq f_i + 1$.□

[Cluster ordering perception] The process p_i perceives that $m_1 \prec m_2$ if at least f_i replicas perceive that $m_1 \prec m_2$. □

[Fault-tolerant cluster ordered (CFO) delivery] For every pair of messages m_1 and m_2, the communication system delivers m_1 before m_2 to p_i if at least f_i replicas in $c(p_i)$ receive m_1 before m_2 and $m_1 \prec m_2$. □

If the CFO delivery rule is not satisfied, p_i considers $m_1 \parallel m_2$.

[Fault-tolerant causally ordered (FCO) delivery] For every pair of messages m_1 and m_2, delivery m_1 before m_2 if every p_i perceives that $m_1 \prec m_2$. □

There is still a case that a replica p_{jh} notifies p_{ik} that $m_2 \prec_{jh} m_1$ even if the correct replica notifies that $m_1 \prec m_2$.

7 Concluding Remarks

The paper discusses how to support the atomic and ordered delivery of message in the group in the presence of process faults.

Acknowledgement

We would like to acknowledge Mr. Hiroaki Higaki, NTT Software lab. for his useful and helpful discussions.

References

[1] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recov-

ery in Database Systems," *Addison-Wesley*, 1987.

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

[3] Birman, K. P., Schiper, A., and Stephenson, P., "Lightweight Causal and Atomic Group Multicast," *ACM TOCS*, Vol.9, No.3, 1991, pp.272-314.

[4] Budhiraja, N., Marzullo, K., Schneider, B. F., and Toueg, S., "The Primary-Backup Approach," *Distributed Computing Systems, ACM Press*, 1994, pp.199-221.

[5] Chandy, K. M. and Lamport, L., "Distributed Snapshots : Determining Global States of Distributed Systems," *ACM TOCS*, Vol.3, No.1, 1985, pp.63–75.

[6] 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.

[7] Fischer, J. M., Nancy, A. L., Michael, S. P., "Impossibility of Distributed Consensus with One Faulty Process," *The Association for Computing Machinery*, Vol.32, No.2, 1985, pp.374–382.

[8] Ezhilchelvan, D. P., Macedo, A. R., and Shrivastava, K. S., "Newtop: A Fault-Tolerant Group Communication Protocol," *IEEE CS Press*, 1995, pp.296–306.

[9] Garcia-Molina, H. and Spauster, A., "Ordered and Reliable Multicast Communication," *ACM TOCS*, Vol.9, No.3, 1991, pp.242–271.

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

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

[12] Lamport, L., Shostak, R., and Pease, M., "The Byzantine Generals Problem," *ACM Trans. Programming Languages and Systems*, Vol.4, No.3, 1982, pp.382–401.

[13] Moser, L. E., Amir, Y., Melliar-Smith, P. M., and Agarwwal, D. A., "Extended Virtual Synchrony," *Proc. of the 14th IEEE ICDCS*, 1994, pp.56-65.

[14] Nakamura, A. and Takizawa, M., "Reliable Broadcast Protocol for Selectively Ordering PDUs," *Proc. of the 11th IEEE ICDCS*, 1991, pp.239-246.

[15] Nakamura, A. and Takizawa, M., "Priority-Based Total and Semi-Total Ordering Broadcast Protocols," *Proc. of the 12th IEEE ICDCS*, 1992, pp.178-185.

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

[17] Schneider, B. F., "Replication Management using the State-Machine Approach," *Distributed Computing Systems, ACM Press*, 1993, pp.169-197.

[18] Schneider, B. F., "Byzantine Generals in Action: Implementing Fail-stop Processors," *ACM TOCS*, Vol.2, No.2, 1984, pp.145–154.

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

[20] Takizawa, M., "Cluster Control Protocol for Highly Reliable Broadcast Communication," *Proc. of the IFIP Conf. on Distributed Processing*, 1987, pp.431-445.

[21] Thomas, L. C. and Mukesh, S., "The Delta-4 Distributed Fault-Tolerant Architecture," *Distributed Computing Systems, IEEE CS Press*, 1990, pp.223-247.