Vol. 37 No. 5 Transactions of Information Processing Society of Japan May 1996

Regular Paper

Fault-Tolerant Intra-Group Communication

KENJI SHIMA,† HIROAKI HIGAKI† and MAKOTO TAKIZAWA†

In distributed systems, a group of processes are cooperated to execute an application program. A group is established among multiple processes and only processes in the group communicate with each other. This type of group communication is named intra-group communication. The communication system has to support reliable intra-group communication in the presence of the process fault. In order to tolerate the process fault, each process in the group is replicated into a collection of multiple replicas named a cluster. In this paper, we would like to propose a new intra-group communication protocol which supports the causally ordered delivery of messages for the processes within the group. In addition, the protocol supports the reliable delivery of messages in the presence of Byzantine faults of the processes.

1. Introduction

In distributed systems executing such applications as teleconferences and telemedicines [7], a group of multiple application processes communicate with each other. The application processes in the group require to receive reliably the messages in the causal order [2]. If the communication system provides the group of processes with a reliable and causally ordered delivery of messages in the group, the distributed applications can be easily realized.

There are kinds of distributed systems. The first one is composed of two types of processes, i.e. clients and servers. The clients send request messages to the servers, and the servers execute the requests and send the responses back to the clients. There is another kind of distributed system like computer network systems, multimedia communication systems, and computer supported cooperative work (CSCW) systems, which are composed of processes which autonomously compute and communicate with other processes [6),18]. In these systems, a group of multiple autonomous processes are cooperated to achieve some objectives. Here, it is required to achieve the *intra-group* communication [15)~17),22] where the processes communicate with each other in the group.

The processes in the distributed system may suffer from kinds of faults. An approach towards making the system fault-tolerant is to replicate the processes in the system. In this paper, in order to support the fault-tolerant group communication, each process is replicated into a collection of multiple *replicas*, which is named a *cluster*. Our protocol supports the inter-cluster communication among the replicas in the clusters in order to tolerate Byzantine faults of processes in the group.

By using the group communication service, the application processes can send reliably messages to the others in the group in some delivery order like the *causal order* [3),17]. Takizawa *et al.* [15),16),22] discuss kinds of group communication protocols which can detect and recover from the message loss as long as all the processes are operational. Birman *et al.* [2] and Moser *et al.* [14] discuss how to manage the membership of the group in the presence of process stop-faults. Ezhilchelvan *et al.* [7] discuss a fault-tolerant group communication protocol where even if a process stops by fault, messages sent by the process are eventually delivered to the destinations in the causal order. Higaki [11] designs the inter-cluster communication protocol which tolerates the stop-fault and message loss. But, the protocol can neither tolerate the Byzantine fault nor provide the causally ordered delivery of messages.

A *logical* group is a collection of multiple processes $p_1, \ldots, p_n$. A *group* is composed of multiple *clusters* each of which is a collection of replicas of the process in the logical group. In order to realize the reliable transmission of a message m from p_i to p_j in the presence of the Byzantine fault, each replica in a cluster of p_i sends m to multiple replicas in a cluster of p_j. Moreover, each replica of p_j receives messages from multiple replicas of p_i. The replicas in the cluster may receive messages in different orders. In this paper, we would like to discuss how the clusters communicate with each other in the group and

† Department of Computers and Systems Engineering, Tokyo Denki University

how the replicas support the processes with the reliable and ordered delivery of messages in the presence of the Byzantine faults of the replicas.

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

2. Replication Schemes

We would like to consider how to replicate a process p_i into replicas $p_{i1}, \ldots, p_{il_i}$ $(l_i \geq 1)$. There are two kinds of approaches towards replicating p_i [4),19)]:

(1) *state-machine approach*, and

(2) *primary-backup approach*.

In the state-machine approach (*active replication*) [19)], every replica p_{ij} is modeled as a deterministic finite state machine. That is, every p_{ij} does the same computation by receiving and sending the same messages $(j = 1, \ldots, l_i)$. Even if some replica of p_i is faulty, the computation of p_i can be continued without stopping. If a replica p_{ij} sends a message different from one which the majority of the replicas send, p_{ij} can be considered to be faulty.

In the primary-backup approach (*passive replication*) [4)], there is one *primary* replica p_{i1}. The other replicas $p_{i2}, \ldots, p_{il_i}$ are named *backup* ones. p_{i1} receives and sends messages and computes while no backup replica computes. p_{i1} saves its local state ls_{i1}^k in its local stable storage at the checkpoint ck_{i1}^k. At the same time, p_{i1} sends ls_{i1}^k to all the backup replicas. On receipt of ls_{i1}^k from p_{i1}, every backup replica p_{ij} saves ls_{i1}^k into the stable storage. If p_{i1} is faulty, one backup replica p_{ij} is selected as a new primary replica. p_{ij} starts the computation of p_i from the checkpoint ck_{ij}^k by restoring the state in ls_{ij}^k. It requires a certain time-overhead for rollbacking the replicas.

The state-machine approach implies more redundant processing and communication than the primary backup one because all the replicas do the same computation by sending and receiving the same messages. However, it requires less time-overhead for recovering from the faults, and the computation can be immediately taken over by the other replicas if some replica is faulty. Moreover, the replicated processes might tolerate the Byzantine faults. Therefore, we would like to adopt the state-machine approach in the rest of this paper.

3. Intra-Group Communication

A distributed application program is executed by the cooperation of multiple processes $p_1, \ldots, p_n$ $(n \geq 2)$ communicating with each other by using the communication system. A collection of $p_1, \ldots, p_n$ is referred to as *group* G, written as $G = \langle p_1, \ldots, p_n \rangle$. Each process p_i is modeled as a finite state machine [19)]. A state is transitted when an event occurs. These are three kinds of events: *sending*, *receipt*, and local events. Here, let $s_i(m)$ and $r_i(m)$ denote sending and receipt events of a message m in p_i, respectively. The local events occur when the local operations are computed in the process. The computation of p_i is modeled to be a sequence of events occurring in p_i.

Lamport [12)] defines the *happened-before* relation $\rightarrow$ on the events as follows:

[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) $e_1 = s_i(m)$ and $e_2 = r_j(m)$, or

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

A causal precedence relation $\prec$ among messages [3)] is defined as follows:

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

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 $m_1 \parallel m_2$.

When a process p_i sends a message m, the communication system delivers m to the destination processes in G. If m is received by all the destinations, m is delivered to the application processes. For every pair of messages m_1 and m_2, m_1 is referred to as *delivered before* m_2 iff the communication system delivers m_1 before m_2 to every common destination process of m_1 and m_2 in G.

[Causally ordered delivery] The communication system supports the causally ordered delivery of messages iff for every pair of messages m_1 and m_2, m_1 is delivered before m_2 if $m_1 \prec m_2$. □

In **Fig. 1**, there are three processes p_i, p_j, and p_k. After sending a message m_0 to p_k, p_i sends m_1 to p_j and p_k. p_j sends m_2 to p_k after receiving m_1. Thus, $m_0 \prec m_1 \prec m_2$ is satisfied. If p_k receives m_0, m_1, and m_2 in this order, the

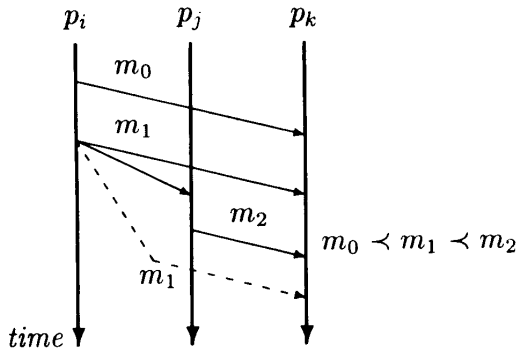

Fig. 1 Causally ordered delivery.

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.

When p_i sends a message m to multiple processes in the group G, m should be delivered to all the destinations.

[Reliable delivery] The communication system *reliably* delivers a message m iff 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. If some process is faulty, no process in the group receives m. Here, processes which are not faulty are *operational*. Even if some faulty processes do not receive m, the operational processes may receive m.

[Operationally reliable delivery] The communication system *operationally* and *reliably* delivers a message m if all the operational destination processes of m receive m. □

m can be retransmitted to a process p_i if p_i does not receive m. If p_i is faulty, m can be sent to p_i after p_i recovers from the fault. Thus, since it takes time to recover from the fault, the response time is increased.

[Fault-tolerant delivery] The communication system *fault-tolerantly* delivers a message m if m is reliably delivered and every destination of m receives m in some *bounded* time after m is sent. □

In the fault-tolerant delivery, m is received by every destination process as if there were no fault. We would like to discuss how to support the fault-tolerant delivery of messages in the presence of the process faults.

4. Inter-Cluster Communication

In order to tolerate the process faults in the group, the processes are replicated to multiple replicas. For a logical group composed of n processes $\langle p_1, \ldots, p_n \rangle$, a *group* G is composed of

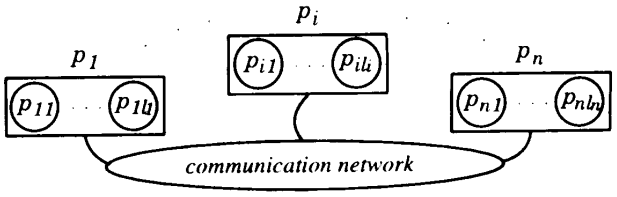

Fig. 2 Group.

n clusters $c(p_1), \ldots, c(p_n)$. Each cluster $c(p_i)$ is a collection of replicas $p_{i1}, \ldots, p_{il_i}$ $(l_i \geq 2f_i+1)$ of p_i $(i = 1, \ldots, n)$ (**Fig. 2**). Every two replicas might not be assigned on the same processor in order to make the faults independent. The replicas communicate with each other by using the communication system. All replicas in $c(p_i)$ behave identically, i.e. each replica is a deterministic finite state machine which receives the same messages, does the same computation, and sends the same messages to support the robustness to the Byzantine faults of processes.

The replica suffering from the Byzantine fault sends incorrect messages, no message, or sends to incorrect destinations. The Byzantine fault of the replica can be detected by comparing the messages sent by the replicas. A message m sent by $c(p_i) = \{p_{i1}, \ldots, p_{il_i}\}$ has to be delivered to the clusters of the destination processes, say $c(p_j) = \{p_{j1}, \ldots, p_{jl_j}\}$. Each replica p_{jh} receives messages from multiple replicas in $c(p_i)$ while p_{jh} sends a message to multiple replicas in another cluster. Some messages may be sent by faulty replicas. Hence, the replicas in $c(p_j)$ have to detect messages sent by the operational replicas in $c(p_i)$. The replicas in $c(p_j)$ have to receive messages from more than $2f_i$ replicas in $c(p_i)$ if at most f_i replicas are faulty. If the replicas in $c(p_j)$ receive the same message m from more than f_i replicas in $c(p_i)$, the replicas can accept m. In addition, the replicas in $c(p_i)$ consider that the faulty replicas have sent messages different from m sent by f_i+1 operational replicas.

5. Fault-Tolerant Ordered Delivery

In the group communication, the messages sent in the logical group of processes $p_1, \ldots, p_n$ are required to be reliably and causally delivered to the application processes. In a group G for the logical group, each process p_i is replicated to be a cluster of l_i $(\geq 2f_i + 1)$ replicas, i.e. $c(p_i) = \{p_{i1}, \ldots, p_{il_i}\}$ $(i = 1. \ldots, n)$ where each p_{ij} is a replica of p_i.

5.1 Assumptions

Faults on the processes are assumed to be Byzantine faults and to be independent in this paper. Due to the processor fault like memory error, 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. We make the following assumption on the number of faulty replicas.

(C1) At most f_i ($\leq l_i$) replicas are faulty at the same time in each cluster $c(p_i)$.

Finally, the communication system is assumed to satisfy the following properties:

(C2) The communication system is *reliable* and *synchronous* [9], i.e. messages are neither lost, contaminated, nor duplicated, and the maximum propagation delay (δ) is predetermined.

(C3) If a process p_i sends p_j a message m_1 before m_2, p_j receives m_1 before m_2. That is, the communication system supports the ordered delivery of messages.

That is, if a replica p_{ik} sends messages to p_{jh}, p_{jh} receives all and only the messages sent by p_{ik} in the sending order.

In addition, we make the following assumptions:

(C4) Each message m has a unique identifier. This is realized by using a process identifier denoted by $m.src$ and a *sequence number* denoted by $m.sn$, given by a process.

(C5) The replicas cannot change the identifier of the message.

If each replica p_{jh} receives two messages m_1 and m_2 from different replicas p_{i1} and p_{i2}, respectively, p_{jh} can decide that m_1 and m_2 are the same messages if $m_1.sn = m_2.sn$ and $m_1.src = m_2.src$. 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. A faulty replica may change a content of m but cannot change $m.sn$ and $m.src$. If p_{jh} does not receive messages from p_{ik} in some predetermined (δ) time units after receiving a message from some replica in $c(p_i)$, p_{jh} considers that p_{jh} receives a *null* message from p_{ik}.

5.2 Fault-tolerant delivery

We would like to consider the fault-tolerant delivery of messages in the group G. Here, suppose that a process p_i sends a message m to p_j in G. That is, the replicas in $c(p_i)$ send m to the replicas in $c(p_j)$ in G by some inter-cluster communication method. p_j accepts m if at least $f_j + 1$ replicas in $c(p_j)$ accept m from $c(p_i)$. If all the destination processes of m accept m, m is fault-tolerantly delivered in G.

We would like to discuss how the clusters in G communicate with each other. Suppose that a process p_i sends a message m to p_j in G. Here, we make a constraint that every replica in $c(p_j)$ receive messages from $c(p_i)$ and decide by itself if the received message is correct. We have to think about which replica in $c(p_i)$ sends m and to which replicas in $c(p_j)$ each replica in $c(p_i)$ sends m. There are four ways for the replicas in the cluster $c(p_i)$ to send m to $c(p_j)$ in G:

(B) each replica p_{ik} in $c(p_i)$ sends m to all replicas $p_{j1}, \ldots, p_{jl_j}$ in $c(p_j)$ for $k = 1, \ldots, l_i$,

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

(MB) each replica p_{ik} in a subset $S(p_i) \subseteq c(p_i)$ sends m to all replicas in $c(p_j)$, and

(MSB) each replica p_{ik} in a subset $T(p_i) \subseteq c(p_i)$ sends m to a subset $K_j(p_{ik}) \subseteq c(p_j)$.

In the first way, each p_{ik} sends m to l_j replicas in $c(p_j)$ ($j \neq i$). $l_i \cdot l_j$ messages are transmitted to deliver m to $c(p_j)$ from $c(p_i)$ in the one-to-one network. In the broadcast network, l_i messages are transmitted to deliver m to $c(p_j)$. This method is named a *broadcast* (B) distribution one.

The second way is that each replica p_{ik} sends m to not necessarily all the replicas in $c(p_j)$ but only a subset $I_j(p_{ik}) \subseteq c(p_j)$. Each replica p_{jh} in $c(p_j)$ has to receive at least $2f_i + 1$ messages from $c(p_i)$ since f_i replicas may be faulty in $c(p_i)$. Hence, $I_j(p_{ik})$ has to satisfy the following constraints;

(1) $I_j(p_{i1}) \cup \cdots \cup I_j(p_{il_i}) = c(p_j)$, where every $I_j(p_{ik}) \neq \phi$, and

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

The total number of messages transmitted from $c(p_i)$ to $c(p_j)$ 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, $l_i - [(2f_i + 1)l_j \ modulo \ l_i]$ 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 is a *selective* broadcast (SB) distribution method.

In the third way, only a subset $S(p_i)$, not necessarily all the replicas, in $c(p_i)$ send m to all

the replicas in $c(p_j)$. Since each replica p_{jh} in $c(p_j)$ is required to receive the messages from more than $2f_i$ replicas in $c(p_i)$, $S(p_i)$ includes more than $2f_i$ replicas in $c(p_i)$. Here, let g_i be $|S(p_i)| \cdot l_i \geq g_i \geq 2f_i + 1$. If $g_i = l_i$, this is the same as the first B method. In the one-to-one network, $g_i \cdot l_j$ messages are transmitted. In the broadcast network, g_i messages are transmitted. This is named a *minimum broadcast* (MB) distribution method.

The fourth way shows the most general way. Here, not necessarily all the replicas in $c(p_i)$ send m to $c(p_j)$ like the SB method and the replicas send m to not necessarily all the replicas in $c(p_j)$ like the MB method. Only g_i replicas in a subset $T(p_i) \subseteq c(p_i)$ send m. Each replica p_{ik} sends m to only a subset $K_j(p_{ik}) \subseteq c(p_j)$. Since each replica p_{jh} in $c(p_j)$ has to receive m from more than $2f_i$ replicas in $c(p_i)$, $K_j(p_{ik})$ has to satisfy the following constraints:

(1) $K_j(p_{i1}) \cup \cdots \cup K_j(p_{il_i}) = c(p_j)$ where $K_j(p_{ik}) = \phi$ if p_{ik} does not send m,

(2) $|\{p_{ik} | p_{jh} \geq K_j(p_{ik})\}| = 2f_i + 1$ for every p_{jh}, and

(3) $g_i = |\{p_{ik} | K_j(p_{ik}) \neq \phi\}| \geq 2f_i + 1$.

Here, we make a constraint that every replica in $T(p_i)$ send m to e_i replicas in $c(p_j)$, i.e. $e_i = (|K_j(p_{i1})| + \cdots + |K_j(p_{il_i})|)/g_i$. Totally $g_i \cdot e_i$ messages are transmitted from $c(p_i)$ to $c(p_j)$ where $g_i \cdot e_i / l_i \geq 2f_i + 1$, $2f_i + 1 \leq g_i \leq l_i$, and $1 \leq e_i \leq l_j$. If $g_i = l_i$, this is the same as the third MB one. The more replicas in $c(p_i)$ send, the less messages the replicas in $c(p_i)$ send. In the broadcast network, g_i messages are transmitted as the MB. This method is named a *minimum selective* broadcast (MSB) distribution one.

Figure 3 shows examples of the B (1) and SB (2) methods where four replicas of p_i ($l_i = 4$) send a message m to five replicas of p_i ($l_j = 6$) for $f_i = 1$. In the B, every replica sends m to all the replicas in $c(p_j)$. Hence, totally $4 \cdot 6 = 24$ messages are transmitted. In the SB, p_{i1} and p_{i2} send five messages, and p_{i3} and p_{i4} send four messages. Each p_{jh} receives three messages from p_i. Totally $(2f_i + 1) \cdot l_j = 3 \cdot 6 = 18$ messages are transmitted. The number of messages in the SB is less than the B.

The replicas which receive messages from other clusters and send messages to other clusters are referred to as *input* and *output* replicas, respectively. In Fig. 3, every replica p_{ik} is an output one. In the MB and MSB methods, g_i output replicas send messages.

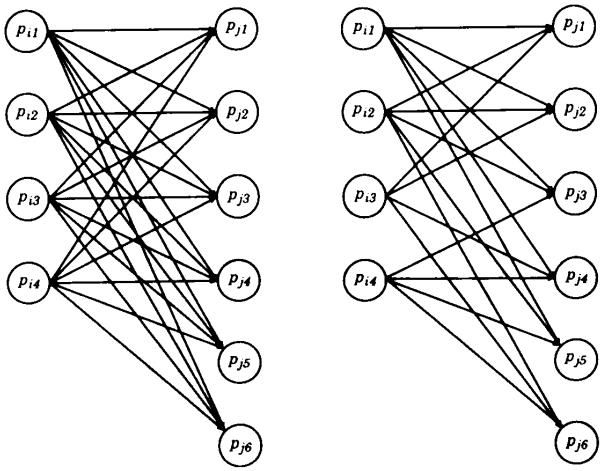

Fig. 3 Inter-cluster communication.

Table 1 Number of messages.

Method	# of messages	#/output	#/input
B	$l_i \cdot l_j$	l_j	l_i
SB	$(2f_i + 1) \cdot l_j$	$(2f_i + 1)l_j/l_i$	$2f_i + 1$
MB	$(2f_i + 1) \cdot l_j$	l_j	$2f_i + 1$
MSB	$(2f_i + 1) \cdot l_j$	$(2f_i + 1)l_j/g_i$	$2f_i + 1$

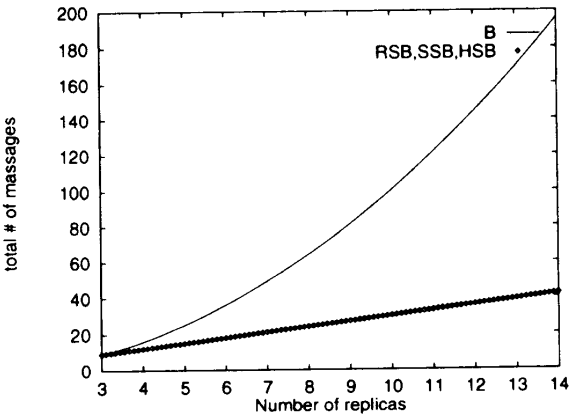

Fig. 4 Total number of messages.

Table 1 shows the number of messages for the distribution methods presented here, where $g_i = (l_i + 2f_i + 1)/2$, $l_i = l_j$, and $f_i = f_j = 1$. Here, *# of messages* shows the total number of messages transmitted from $c(p_i)$ to $c(p_j)$, *#/output* and *#/input* show the numbers of messages sent and received by each output repica p_{ik} and input replica p_{jh}, respectively. From Table 1, in the SB, MB, and MSB methods, each replica receives less number of messages than the B method. **Figure 4** shows the total number of messages transmitted from $c(p_i)$ to $c(p_j)$ for l_i where $l_i = l_j$ and $f_i = f_j = 1$.

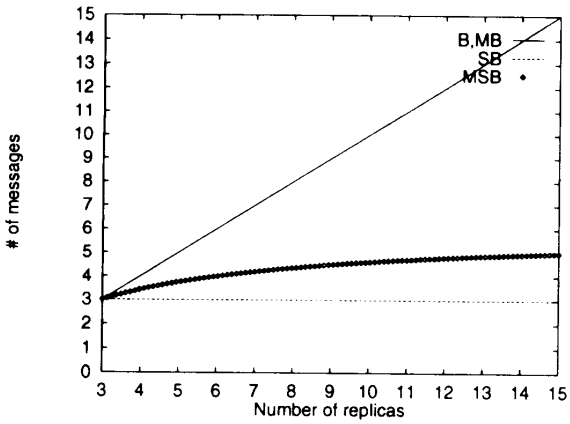

Fig. 5 Number of messages sent by output replicas.

From Fig. 4, in the SB, MB, and MSB methods, the same number of messages are transmitted, but less number of message are transmitted than the B. **Figure 5** shows #/output for l_i. In the SB, each output replica sends the smallest number of messages. Therefore, the SB method is the best one.

5.3 Ordered delivery

The communication system has to deliver messages to the destination processes in some well-defined order. The replicas in a cluster $c(p_j)$ may receive messages in different orders due to the communication delay and the Byzantine fault of the replicas.

[Simply ordered delivery] The communication system supports a process p_j with the *simply ordered delivery* of messages iff for every pair of messages m_1 and m_2,

(1) if m_1 and m_2 are sent by the same process, m_1 is delivered before m_2 to p_j if $m_1.sn < m_2.sn$,

(2) otherwise, m_1 is delivered to p_j before m_2 if more than f_j operational replicas in $c(p_j)$ receive m_1 before m_2. □

The communication system can deliver m_1 and m_2 in any order if m_1 and m_2 do not satisfy the simply ordered delivery rule.

The communication system has to support the causally ordered delivery [3] of messages for the processes in the group G. First, we would like to discuss the precedence relation among the messages. As presented in the papers [17),21)], each message m sent by p_{ik} carries the confirmation field ack_{jh} which denotes the sequence number of the message which p_{ik} knows p_j expects to receive next from p_h $(h = 1, \ldots, n)$. The messages can be causally ordered as follows [17)]:

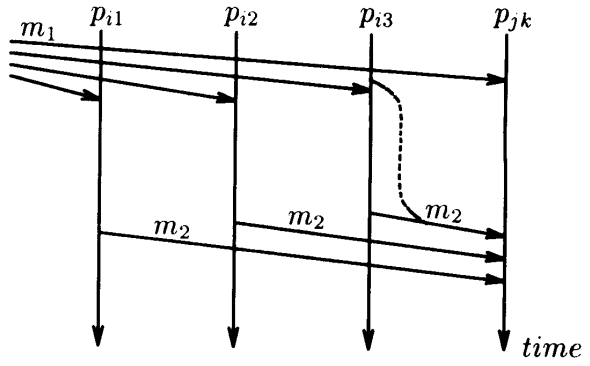

Fig. 6 Causally ordered delivery.

[Causally oredering rule] For every pair of messages m_1 and m_2, m_1 *causally precedes* 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. □

It is straightforward that the following property holds from the causally ordering rule.

[Proposition] On receipt of messages m_1 and m_2, each replica p_{ij} can decide whether $m_1 \prec m_2$, $m_2 \prec m_1$, or $m_1 || m_2$. □

It is noted that the faulty replica can change the confirmation fields *ack* in the messages while it cannot change the sequence numbers. This means that the faulty replica may send the incorrect precedence information to other replicas. For example, in **Fig. 6**, three replicas p_{i1}, p_{i2}, and p_{i3} in $c(p_i)$ send m_2 to p_{jh} in $c(p_j)$ after receiving m_1. That is, $m_1 \prec m_2$. Here, suppose that p_{i3} is faulty while p_{i1} and p_{i2} are operational. p_{i3} sends m_2 with the incorrect causality information $m_1 \,||\, m_2$ while p_{i1} and p_{i2} send m_2 with $m_1 \prec m_2$.

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

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

That is, each replica in $c(p_j)$ decides "$m_1 \prec m_2$"if more than f_i replicas in $c(p_i)$ notify that m_1 causally precedes m_2, i.e. $m_1 \prec m_2$.

[Process perception] Each process p_j perceives that $m_1 \prec m_2$ if at least f_j+1 operational replicas in $c(p_j)$ perceive that $m_1 \prec m_2$. □

In Fig. 6, p_{jh} receives m_2 from p_{i1}, p_{i2}, and p_{i3}. p_{i1} and p_{i2} notify p_{jh} of "$m_1 \prec m_2$" by m_2. p_{i3}

notifies p_{jh} of "$m_1 || m_2$" by m_2 while p_{jh} sends m_2 after receiving m_1, i.e. p_{i3} is faulty. Here, suppose that $f_i = 1$. p_{jh} decides "$m_1 \prec m_2$" by the replica perception rule because $f_i + 1$ (= 2) replicas notifying p_{jh} of "$m_1 \prec m_2$". If the process perception rule is not satisfied, p_i considers $m_1 || m_2$.

[Causally ordered delivery] The communication system supports the causally ordered delivery of messages iff m_1 is delivered before m_2 to every common destination of m_1 and m_2 if $m_1 \prec m_2$ for every pair of messages m_1 and m_2. □

There is still a case that some replica sends m_2 after receiving m_1 and others send m_2 before receiving m_1 in $c(p_j)$. If the process perception rule is not satisfied for m_1 and m_2 in p_i, p_i perceives $m_1 || m_2$. If some replica p_{jh} is faulty in $c(p_j)$, one process p_i may perceive "$m_1 || m_2$" while the other process p_l perceives "$m_1 \prec m_2$" since p_{jh} notifies the replicas in $c(p_i)$ and $c(p_l)$ of the different ordering information. Our method guarantees that m_1 is delivered before m_2 if $m_1 \prec m_2$. **Figure 7** shows the communication among two clusters $c(p_i)$ and $c(p_j)$, each of which includes three replicas. Suppose that the replicas p_{i1} and p_{i3} are operational but p_{i2} is faulty. p_{i1} sends m_2 before receiving m_1 and p_{i3} sends m_2 after receiving m_1. p_{i2} sends m_2 to $c(p_j)$ where p_{i2} notifies p_{j1} and p_{j2} of $m_1 \prec m_2$ and p_{j3} of $m_1 || m_2$. By the process perception rule, $c(p_j)$ decides that $m_1 \prec m_2$. This shows that m_1 is delivered before m_2 if $m_1 \prec m_2$ but m_1 may not causally precede m_2 if m_1 is delivered before m_2. Thus, it is straightforward for the following theorem to hold.

[Theorem] For every pair of messages m_1 and m_2, m_1 is delivered before m_2 to every common destination process of m_1 and m_2 if $m_1 \prec m_2$. □

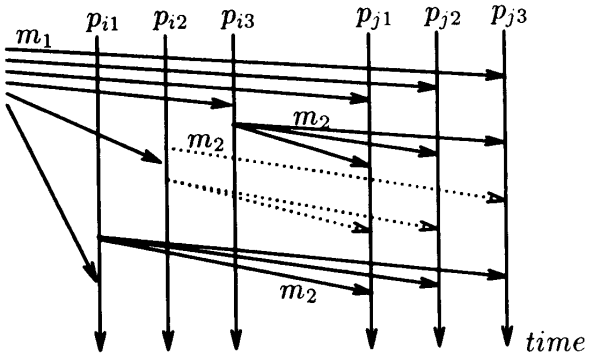

Fig. 7 Causally ordered delivery.

6. Concluding Remarks

This paper discusses how to support the fault-tolerant and causally ordered delivery of messages in the group in the presence of the Byzantine faults of processes. Each process is modeled to be a deterministic finit state machine and is actively replicated to a set of multiple replicas, named a cluster. In this paper, we have discussed protocols for the inter-cluster communication in the group and shown the evaluation of them. The faulty replicas may send different messages or no messages and may include the incorrect ordering information on the messages. We have discussed how to treat the faults of the unreliable and incorrectly ordered delivery of messages.

References

1) Bernstein, P.A., Hadzilacos, V. and Goodman, N.: *Concurrency Control and Recovery 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, pp.47–76 (1987).
3) Birman, K.P., Schiper, A. and Stephenson, P.: Lightweight Causal and Atomic Group Multicast, *ACM TOCS*, Vol.9, No.3, pp.272–314 (1991).
4) Budhiraja, N., Marzullo, K., Schneider, B.F. and Toueg, S.: The Primary-Backup Approach, *Distributed Computing Systems*, ACM Press, pp.199–221 (1994).
5) Chandy, K.M. and Lamport, L.: Distributed Snapshots: Determining Global States of Distributed Systems, *ACM TOCS*, Vol.3, No.1, pp.63–75 (1985).
6) Cooper, E.C.: Replicated Distributed Programs, *Proc. 10th ACM Symp. on Operating Systems Principles*, pp.63–78 (1985).
7) Ellis, C.A., Gibbs, S.J. and Reln, G.L.: Groupware: Some Issues and Experiences, *Comm. ACM*, Vol.34, No.1, pp.39–58 (1991).
8) Ezhilchelvan, D.P., Macedo, A.R. and Shrivastava, K.S.: *Newtop: A Fault-Tolerant Group Communication Protocol*, IEEE CS Press, pp.296–306 (1995).
9) Fischer, J.M., Nancy, A.L. and Michael, S.P.: Impossibility of Distributed Consensus with One Faulty Process, *ACM TOCS*, Vol.32, No.2, pp.374–382 (1985).
10) Garcia-Molina, H. and Spauster, A.: Ordered and Reliable Multicast Communication, *ACM TOCS*, Vol.9, No.3, pp.242–271 (1991).
11) Higaki, H. and Soneoka, T.: Group-to-Group Communications for Fault-Tolerance in Dis-

tributed Systems, *IEICE Trans.on Information and Systems*, Vol.E76-D, No.11, pp.1348–1357 (1993).
12) Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System, *Comm. ACM*, Vol.21, No.7, pp.558–565 (1978).
13) Lamport, L., Shostak, R. and Pease, M.: The Byzantine Generals Problem, *ACM Trans. Prog. Lang. Syst.*, Vol.4, No.3, pp.382–401 (1982).
14) Moser, L.E., Amir, Y., Melliar-Smith, P.M. and Agarwal, D.A.: Extended Virtual Synchrony, *Proc. 14th IEEE ICDCS*, pp.56–65 (1994).
15) Nakamura, A. and Takizawa, M.: Reliable Broadcast Protocol for Selectively Ordering PDUs, *Proc. 11th IEEE ICDCS*, pp.239–246 (1991).
16) Nakamura, A. and Takizawa, M.: Priority-Based Total and Semi-Total Ordering Broadcast Protocols, *Proc. 12th IEEE ICDCS*, pp.178–185 (1992).
17) Nakamura, A. and Takizawa, M.: Causally Ordering Broadcast Protocol, *Proc. 14th IEEE ICDCS*, pp.48–55 (1994).
18) Powell, D., Chereque, M. and Drackley, D.: Fault-Tolerance in Delta-4, *ACM Operating System Review*, Vol.25, No.2, pp.122–125 (1991).
19) Schneider, B.F.: Replication Management using the State-Machine Approach, *Distributed Computing Systems*, ACM Press, pp.169–197 (1993).
20) Schneider, B.F.: Byzantine Generals in Action: Implementing Fail-stop Processors, *ACM TOCS*, Vol.2, No.2, pp.145–154 (1984).
21) Tachikawa, T. and Takizawa, M.: Selective Total-Ordering Group Communication on Single High-Speed Channel, *Proc. IEEE ICNP-94*, pp.212–219 (1994).
22) Takizawa, M.: Cluster Control Protocol for Highly Reliable Broadcast Communication, *Proc. IFIP Conf. on Distributed Processing*, pp.431–445 (1987).

(Received October 2, 1995)
(Accepted March 12, 1996)

Kenji Shima was born in Japan on Jan 13, 1972. He received his B.E. degree from the Dept. of Computers and Systems Enginering, Tokyo Denki University in 1995. He is now a graduate student of the master course in the Dept. of Computers and Systems Engineering, Tokyo Denki University. His research interest includes fault-tolerant distributed systems, computer networks, and communication protocols.

Hiroaki Higaki was born in Tokyo, Japan, on April 6, 1967. He received the B.E. degree from the Department of Mathematical Engineering and Information Physics, the University of Tokyo in 1990. From 1990 to 1996, he was in NTT Software Laboratories. Since 1996, he is in the Faculty of Science and Engineering, Tokyo Denki University. His research interest includes distributed algorithms, distributed operating systems and computer network protocols. He received IPSJ Convention Award in 1995. He is a member of ACM and IEICE.

Makoto Takizawa was born in 1950. He received his B.E. and M.E. degrees in Applied Physics from Tohoku University, in 1973 and 1975, respectively. He received his D.E. in Computer Science from Tohoku University in 1983. From 1975 to 1986, he worked for Japan Information Processing Developing Center (JIPDEC) supported by the MITI. He is currently a Professor of the Department of Computers and Systems Engineering, Tokyo Denki University since 1986. From 1989 to 1990, he was a visiting professor of the GMD-IPSI, Germany. He is also a regular visiting professor of Keele University, England since 1990. He was a vice-chair of IEEE ICDCS, 1994 and serves on the program committees of many international conferences. 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.